Pós-Graduação em Ciência da Computação

KNOWING: UM MODELO PARA GARANTIA DE CONSISTÊNCIA DOS DADOS EM SISTEMAS DE BANCO DE DADOS RELACIONAIS EM NUVEM

Por

ELYDA LAISA SOARES XAVIER FREITAS

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, MARÇO/2014

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ELYDA LAISA SOARES XAVIER FREITAS

KNOWING: UM MODELO PARA GARANTIA DE CONSISTÊNCIA DOS DADOS EM SISTEMAS DE BANCO DE DADOS RELACIONAIS EM NUVEM

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: FERNANDO DA FONSECA DE SOUZA

RECIFE, MARCO/2014

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Freitas, Elyda Laisa Soares Xavier
Knowing: um modelo para garantia de consistência dos dados em sistemas de banco de dados relacionais em nuvem / Elyda Laisa Soares Xavier Freitas. - Recife: O Autor, 2014.
111 p., fig., quadros

Orientador: Fernando da Fonseca de Souza.
Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2014.

Inclui referências e apêndice.

1. Ciência da Computação. 2. Banco de dados. 3. Computação em nuvem. I. Souza, Fernando da Fonseca de (orientador). II. Título.

004 CDD (23. ed.) MEI2014 – 041

Dissertação de Mestrado apresentada **por Elyda Laisa Soares Xavier Freitas** à Pós Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Knowing: Um Modelo para Garantia de Consistência dos Dados em Sistemas de Banco de Dados Relacionais em Nuvem**" orientada pelo **Prof. Fernando da Fonseca de Souza** e aprovada pela Banca Examinadora formada pelos professores:

_____

Profa. Ana Carolina Brandão Salgado
Centro de Informática / UFPE

_____

Profa. Maria Lencastre Pinheiro de Menezes Cruz
Escola Politécnica de Pernambuco / UPE

_____

Prof. Fernando da Fonseca de Souza
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 26 de fevereiro de 2014.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

AGRADECIMENTOS

Em primeiro lugar, eu gostaria de agradecer a Deus. Eu creio que foi o Senhor, meu Deus, que me concedeu essa oportunidade de me tornar Mestre, creio que foi Ele que me capacitou, que me deu forças pra seguir e chegar até aqui. Sem Ti, Senhor, não sou nada. Não tenho capacidade de fazer nada sozinha.

Durante o este trabalho grandes mudanças aconteceram na minha vida. E eu gostaria de agradecer a todo mundo que seguiu ao meu lado.

Obrigada, D.Edna, minha mãe, por fazer tudo quanto é possível para que as coisas deem certo pra mim. Pelo cuidado, preocupação e, principalmente, pelo amor que a senhora dedica a mim. Obrigada, Sr. João Eudes, meu pai, por possibilitar meu crescimento pessoal e acadêmico, por meio do seu auxílio sempre presente e pelo amor que o senhor tem por mim.

Meu amor, meu marido, Diógenes Ricardo. Obrigada por me fazer a mulher mais feliz do mundo todos os dias. Estarei até a eternidade contigo.

Minha irmã, Erika, e meu sobrinho, Eder: é bom demais saber que tenho vocês. Obrigada por me apoiarem, por se preocuparem comigo. Sinto muita falta de vocês, de estar mais tempo em casa, de ouvir a voz de Ederzinho me chamando o tempo todo: ô, tia. =)

À minha família, que é meu alicerce nos momentos de dificuldade e minha alegria em todo o tempo.

Ao Professor Fernando: Obrigada pela confiança no meu trabalho, pelas conversas sempre agradáveis. Obrigada porque, de fato, fizemos uma parceria durante a caminhada. Desejo trabalhar com o senhor muitas vezes mais.

Aos professores que me apoiaram na caminhada do mestrado: Almir Moura, que me orientou e incentivou no pré-projeto; À professora Ana Carolina, que nos repassou seu vasto conhecimento com tanta simplicidade e carinho, e que estava sempre disposta a tirar dúvidas e ajudar. E, por fim, por aceitar participar da minha banca. Obrigada, Carol; À professora Bernadette Loscio,

pelas palavras de incentivo e pelo apoio; e a todos do Centro de Informática o meu “muito obrigada”.

Ao meu pastor, Ary, que, mesmo com a distância se preocupou comigo e com a minha vida espiritual. Glória a Deus, que o colocou em nossas vidas.

A cada um dos meus irmãos da 1ª Igreja Evangélica Congregacional de Caruaru: obrigada pelas orações, pela comunhão, pela preocupação. Gostaria de estar aí todos os dias, como antes, mas Deus sabe todas as coisas e a mim, resta esperar nEle.

Aos amigos que fiz e aos que mantive, os quais continuam meus amigos mesmo quando eu sumo. Obrigada!!

RESUMO

Bancos de Dados em Nuvem permitem armazenar grandes volumes de dados e crescer quase indefinidamente. Essa característica, alcançada devido à distribuição dos dados, impõe ao sistema de banco de dados algumas restrições, uma vez que não é possível atingir em um sistema distribuído, e ao mesmo tempo, as desejáveis características de Consistência, Disponibilidade e Tolerância à Partição (em caso de falha), conforme determina o Teorema CAP. Desse modo, diversos sistemas de bancos de dados em nuvem têm optado por relaxar a garantia de consistência, dando prioridade à disponibilidade do serviço. Para grande parte das aplicações, no entanto, inconsistências nos dados podem levar a transtornos imensuráveis, como no caso de uma aplicação bancária com valores equivocados. Estudos recentes mostram a necessidade de investigar e explorar novas opções, as quais possibilitem a garantia de consistência nos sistemas de banco de dados em nuvem. Diante do exposto, este trabalho apresenta o *Knowing*, um modelo de consistência de dados para sistemas de banco de dados relacionais em nuvem, o qual se utiliza do conhecimento do usuário sobre a aplicação para definir quais dados necessitam de garantia de consistência forte e quais não. Os dados com garantia de consistência são tratados por meio da atualização ansiosa. Neste caso, uma arquitetura adaptada à nuvem, a qual se utiliza de comunicação em grupo, foi projetada. Para dados que não necessitam de consistência forte, técnicas de consistência eventual poderão ser utilizadas. Um protótipo foi desenvolvido com a finalidade de avaliar o modelo criado por meio de experimentos. Estes, bem como sua análise, foram também apresentados.

Palavras-Chave: Banco de Dados como Serviço. Consistência de Dados. Banco de Dados em Nuvem.

ABSTRACT

Cloud databases allow storing large volumes of data and grow almost indefinitely. This feature, achieved due to the data distribution, imposes on the database system some restrictions, since the desirable characteristics of Consistency, Availability, and Partition tolerance cannot be reached in a distributed system at the same time (in case of failure), as stated by the CAP Theorem. Thus, many systems in cloud databases have chosen to relax the consistency guarantee, giving priority to service availability. For most applications, however, data inconsistencies can lead to heavy losses, e.g. a banking application with wrong values. Recent studies show the need to investigate and explore new options, which allow guaranteeing consistency in cloud database systems. This work presents Knowing, a data consistency model focusing on relational cloud database systems, which takes into consideration the user's knowledge about the application to define which data needs strong consistency guarantees and which data does not need. Data with strong consistency guarantees are handled by eager update. In this case, an adapted architecture to cloud computing, that uses group communication, was designed. Data that do not require strong consistency can be treated with eventual consistency techniques. A prototype was developed in order to evaluate the model by means of some experiments. These, as well as their analyses were also presented.

Keywords*: Database as a Service. Data Consistency. Cloud Databases.*

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

O uso extensivo da Tecnologia da Informação (TI) já é parte da rotina de pequenas e grandes empresas. Em face disso, um recurso tem se tornado cada vez mais valioso dentro das organizações: seus bancos de dados (BD). É por meio dos BD que podem ser extraídas diversas informações valiosas, sejam elas básicas ou extremamente complexas - obtidas por meio da mineração de dados, cujo objetivo é descobrir informações úteis ocultas em grandes bancos de dados (LIN et al., 2011).

Por permitir o armazenamento de dados de maneira organizada, segura e confiável, os Sistemas de Gerenciamento de Banco de Dados (SGBD) são elementos importantes para se obter de forma eficaz as melhores informações a partir de um conjunto de dados.

Para gerenciar tais SGBD, lidando com possíveis falhas e melhorias, faz-se necessário o trabalho de profissionais especializados: os administradores de banco de dados, ou DBA - *Database Administrators* em inglês (ALZAIN e PARDEDE, 2011).

Concomitantemente, nos últimos anos, uma área de estudo de TI tem se destacado: a Computação em Nuvem. Segundo Foster et al. (2008), a Computação em Nuvem:

> É um paradigma de computação distribuída de larga escala (...) no qual um conjunto gerenciado de recursos computacionais, armazenamento, plataformas e serviços abstraídos, virtualizados e dinamicamente escaláveis são entregues por demanda a clientes externos por meio da Internet.

Nesse modelo, o provedor (geralmente uma empresa) é o detentor do recurso e o concede conforme a necessidade do usuário. Ademais, o pagamento é realizado de acordo com a quantidade de recursos alocados, que é medida de acordo com o tipo de serviço oferecido (*software, hardware*, plataforma de desenvolvimento).

O SGBD é um dos muitos serviços que podem ser disponibilizados na nuvem. Conhecido como DBaaS, do inglês *Database as a Service* (Banco de

Dados como Serviço), este novo paradigma tem recebido a atenção tanto da academia quanto de grandes empresas, como é o caso da Amazon[1], Oracle[2] e do Google[3]. Segundo Hsieh et al. (2011), o tamanho de conjuntos de dados armazenados em nuvem tem crescido rapidamente.

No DBaaS, um provedor de serviços terceirizado hospeda o banco de dados “como um serviço”, provendo aos usuários mecanismos para criar, armazenar e acessar seus bancos de dados em um ambiente externo (HACIGÜMÜS et al., 2002). Em outras palavras, o provedor de serviços disponibiliza todo o ambiente necessário para que os usuários criem e mantenham seus bancos de dados, os quais são acessados por meio da Internet. Desse modo, o usuário não precisa preocupar-se com questões como a aquisição de servidores para hospedagem do banco de dados, instalação de *software* e do próprio SBGD; e até mesmo com tarefas de gerenciamento, tais como otimização de desempenho e balanceamento de carga. Todas essas tarefas passam a ser de responsabilidade do provedor.

A Figura 1.1, a seguir, destaca os recursos disponibilizados em um Banco de Dados como Serviço:

Figura 1.1 - Recursos oferecidos pelos provedores de Bancos de Dados como Serviço.

Fonte: A Autora.

Para dar suporte aos inquilinos, os provedores de serviço dispõem de enormes *datacenters*. Em geral, na Computação em Nuvem, esses *datacenters*

[1] http://aws.amazon.com/pt/rds/
[2] https://cloud.oracle.com
[3] https://developers.google.com/cloud-sql/?hl=pt-BR

são formados por centenas de máquinas que, organizadas em *clusters*, atingem uma alta capacidade de processamento.

A distribuição, inerente ao DBaaS, permite ao banco de dados armazenar grandes volumes de dados e crescer quase indefinidamente – aos limites da capacidade de armazenamento do *datacenter* hospedeiro. No entanto, essa característica impõe ao sistema de banco de dados algumas restrições.

A seção seguinte, portanto, apresenta a motivação para o desenvolvimento do presente trabalho.

1.1. Motivação

Conforme provado por Gilbert e Lynch (2002), não é possível atingir em um sistema distribuído, e ao mesmo tempo, as desejáveis características de Consistência, Disponibilidade e Tolerância à Partição (em caso de falha). Tal proposição é conhecida como Teorema CAP.

De acordo com Öszu e Valduriez (2011), "*A consistência de uma transação é simplesmente sua corretude*". Deste modo, diversos sistemas de bancos de dados em nuvem têm optado por relaxar a garantia de consistência, dando prioridade à disponibilidade do serviço (ZHAO et al., 2012).

Para alguns tipos de aplicações, como as redes sociais, tal opção é perfeitamente válida, uma vez que não há dados críticos armazenados. No Twitter[4], por exemplo, não causa problemas se uma publicação não for imediatamente compartilhada para todos os seguidores de um usuário.

No entanto, para grande parte das aplicações, inconsistências nos dados podem levar a transtornos imensuráveis, como no caso de uma aplicação bancária com valores equivocados. Desse modo, verifica-se que, em função do relaxamento da consistência, diversos sistemas de BD em nuvem não estão aptos a receber qualquer tipo de aplicação.

[4] https://twitter.com/. Acesso: 14/03/2014.

Entretanto, o relaxamento da consistência não é a única possibilidade: A consistência, assim como a disponibilidade, pode ser examinada e tratada em diferentes níveis, conforme explica Brewer (2012): "*A disponibilidade é obviamente contínua de 0 a 100 por cento, mas há também muitos níveis de consistência*".

Desta forma, será apresentada, na seção seguinte, a pergunta de pesquisa do presente trabalho, que trata exatamente do problema de consistência em Banco de Dados em Nuvem.

1.2. Problema de Pesquisa

Diante do exposto anteriormente, torna-se perceptível a necessidade de explorar as diferentes nuances da consistência de um Banco de Dados em Nuvem, ao invés de simplesmente relaxá-la.

Sendo assim, o seguinte problema de pesquisa foi formulado: **Como ajudar a garantir a consistência dos dados armazenados em bancos de dados relacionais em nuvem a fim de permitir a utilização do modelo de DBaaS por diferentes tipos de aplicações[5]?**

1.3. Objetivos

Para responder à pergunta de pesquisa, foram definidos os seguintes objetivos, divididos em geral e específicos:

1.3.1. Geral

Propor um modelo que ajude a garantir a consistência de dados em sistemas de banco de dados relacionais em nuvem, no qual o usuário[6] possa tomar decisões sobre a consistência dos dados.

[5] Isto é, aplicações que necessitam de garantia de consistência e aquelas que não necessitam.
[6] O usuário, neste contexto, é o responsável pelo projeto do banco de dados.

1.3.2. Específicos

- Definir um modelo de consistência voltado para o contexto de BD em nuvem;

- Definir uma arquitetura para dar suporte ao modelo criado;

- Desenvolver um protótipo com um mecanismo que implemente o modelo proposto; e

- Avaliar o mecanismo por meio de experimentos.

1.4. Justificativa

Dez anos após a concepção do Teorema CAP, Brewer (2012) afirma, em seu trabalho intitulado "*CAP Twelve Years Later: How the 'Rules' Have Changed*" (CAP Doze Anos Depois: Como as "Regras" Mudaram), que a compreensão desse teorema foi excessivamente simplificada, levando os projetistas a escolherem indiscriminadamente duas das proposições (como, por exemplo, tolerância à partição e disponibilidade; ou consistência e disponibilidade, apenas). Brewer (2012) recomenda, ainda, que a consistência não seja "cegamente" sacrificada.

Visão semelhante é apresentada por Daniel Abadi (ABADI, 2012), que afirma que o teorema CAP foi mal compreendido, tendo sido ele, inclusive, uma das maiores influências para o surgimento de sistemas com a consistência de dados reduzida nos anos seguintes à sua prova. Mediante essas afirmações, mostra-se latente a necessidade de investigar e explorar novas opções, as quais auxiliem na garantia de consistência nos sistemas de banco de dados em nuvem.

Este trabalho apresenta, portanto, alternativas ajudam a garantir a consistência de dados e que já estão ratificadas na academia, mas ainda não foram experimentadas em um ambiente como a nuvem. As devidas adaptações para esse ambiente serão realizadas.

A arquitetura apresentada poderá servir de ponto de partida para o desenvolvimento de um Sistema de Gerenciamento de Banco de Dados a ser hospedado na nuvem e cujo público-alvo é composto por usuários que necessitam de garantia de consistência no tratamento dos dados.

1.5. Contribuições Esperadas

Espera-se, ao final deste trabalho, alcançar as seguintes contribuições:

- Apresentação de um estudo abrangente dos temas Banco de Dados como Serviço e Consistência de Dados, de modo a facilitar o desenvolvimento de futuros trabalhos dentro desses tópicos;

- Utilização de abordagens de replicação e propagação de dados reconhecidas e largamente estudadas em um ambiente diferente daquele proposto inicialmente para as mesmas, com as devidas adaptações; e

- Desenvolvimento de um modelo que explore os diferentes níveis da consistência de dados, conforme proposto por Brewer (2012).

1.6. Metodologia

Segundo Kahlmeyer-Mertens et al. (2007), a metodologia científica "*se propõe a definir regras e procedimentos que darão segurança e validade ao exercício de conhecer, tendo a pesquisa presente nesse processo*". Para isso, apresenta-se, a seguir, a classificação desta pesquisa, de acordo com Marconi e Lakatos (2000).

1.6.1. Natureza da Pesquisa

Esta pesquisa pode ser classificada como aplicada, uma vez que visa gerar conhecimentos para aplicação prática, dirigidos à solução de problemas específicos (PINHEIRO, 2010).

1.6.2. Quanto aos Fins

Quanto aos fins, a presente pesquisa pode ser classificada como exploratória e explicativa. A pesquisa exploratória visa conhecer inicialmente o tema de estudo. Pois, ainda segundo Pinheiro (2010), a pesquisa exploratória "*visa proporcionar maior familiaridade com o problema com vistas a torná-lo*

explícito ou a construir hipóteses". Neste trabalho, será realizada através da revisão da literatura. Já a pesquisa explicativa, de acordo com Gil (2002), "*tem como preocupação central identificar os fatores que determinam ou que contribuem para a ocorrência dos fenômenos*". Ainda segundo o autor, esse tipo de pesquisa, em geral, vale-se do método experimental.

1.6.3. Quanto aos Meios

Quanto aos meios, esta pesquisa está classificada como experimental, uma vez que segundo Pinheiro (2010), "*a pesquisa experimental ocorre quando se determina um objeto de estudo, selecionando-se as variáveis que seriam capazes de influenciá-lo, definindo-se as formas de controle e de observação dos efeitos que a variável produz no objeto*".

1.6.4. Quanto à Forma de Abordagem

Por fim, quanto à forma de abordagem do problema, a pesquisa classifica-se como qualitativa, pois como afirma Malhotra (2006), a pesquisa qualitativa "*é uma metodologia de pesquisa exploratória e não-estruturada que se baseia em pequenas amostras com o objetivo de prover percepções e compreensão do problema*".

1.6.5. Descrição

Esta seção tem por objetivo descrever brevemente todas as etapas seguidas para o desenvolvimento do presente trabalho, desde a revisão de literatura à análise dos experimentos.

Para produzir o referencial teórico desta pesquisa foram realizadas consultas em livros conceituados relacionados aos temas em discussão, a saber: *Database as a Service* e Consistência de Dados. Adicionalmente, foram consultados artigos disponíveis em bibliotecas científicas, tais como *Association for Computing Machinery* (ACM)[7], *Scientific Eletronic Library Online* (SciELO)[8] e

[7] http://dl.acm.org/. Acesso: 23/01/2014
[8] www.scielo.org/. Acesso: 23/01/2014

Institute of Electrical and Electronic Engineers (IEEE Xplore)[9], explorando, especialmente, artigos publicados nos últimos 5 (cinco) anos.

Juntamente à revisão de literatura, foi realizado um estudo a fim de analisar trabalhos a este relacionados. A revisão de literatura somada a esse estudo permite conhecer mais profundamente os temas em questão, possibilitando a compreensão de seus conceitos e da sua atual conjuntura.

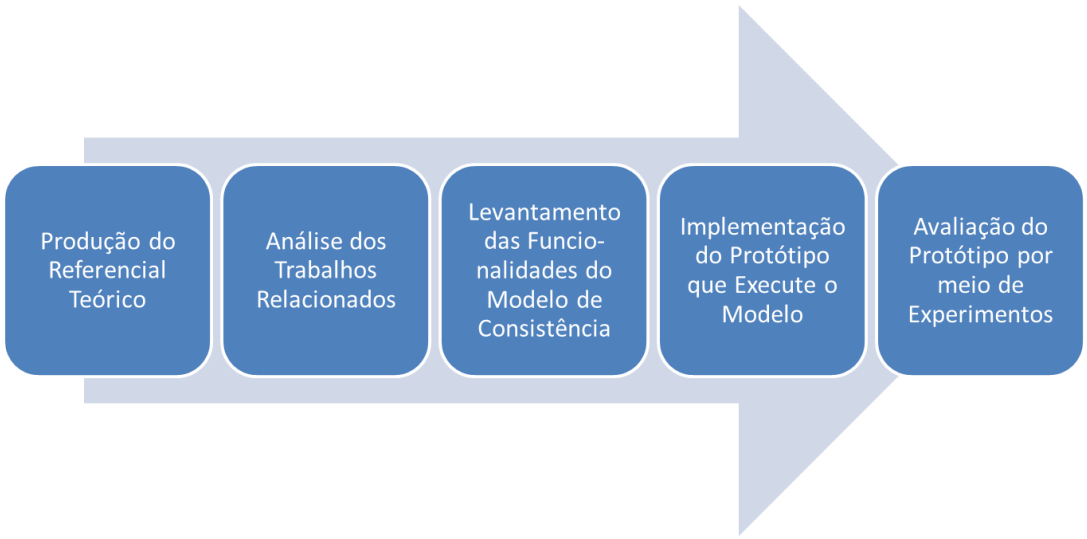
De posse desses conhecimentos, foram levantadas as funcionalidades do modelo de consistência proposto, que consideram os objetivos descritos. Um protótipo deve implementar o modelo proposto.

Por fim, o protótipo foi testado a fim de que sejam verificadas suas capacidades. Os testes foram realizados por meio de consultas SQL executadas simultaneamente a operações de escrita, o que permitirá avaliar a garantia de consistência no ambiente distribuído.

O fluxo de atividades descrito pode ser resumido segundo a

Figura 1.2:

Figura 1.2 - Fluxo de atividades executado para realização da pesquisa.



Fonte: A Autora.

[9] http://ieeexplore.ieee.org/. 23/01/2014.

1.7. Organização do Trabalho

Neste capítulo, foi realizada uma breve introdução aos temas deste trabalho: *Database as a Service* e Consistência de Dados. Ainda neste capítulo, o problema de pesquisa foi apresentado, bem como os objetivos gerais e específicos para responder à pergunta de pesquisa. Além disso, foram apresentadas a justificativa e motivação para realização deste trabalho, descrita a metodologia utilizada e destacadas as contribuições esperadas.

O restante do trabalho está organizado da seguinte forma:

- Capítulo 2 – Referencial Teórico: Expõe o referencial teórico, que aborda de forma mais abrangente os temas *Database as a Service* e Consistência de Dados, além de tratar trabalhos relacionados a este.

- Capítulo 3 - *Knowing*: Um Modelo para Ajudar a Garantir a Consistência de Dados Relacionais em Nuvem: Apresenta o modelo *Knowing*, por meio da explanação de sua arquitetura.

- Capítulo 4 - Um Protótipo para Aplicação do Modelo *Knowing*: Descreve os detalhes da implementação do protótipo.

- Capítulo 5 – Experimentos: Descreve as avaliações realizadas bem como a análise dos resultados.

- Capítulo 6 – Considerações Finais: Expõe as conclusões sobre os estudos e a proposta de trabalhos futuros.

- Apêndice A – Resultado das Consultas: Apresenta o resultado das consultas executadas durante o experimento.

Por fim, são destacadas as referências utilizadas na composição da revisão de literatura e na pesquisa.

2. REFERENCIAL TEÓRICO

A seguir, serão abordados, com apoio da literatura, os principais temas que formam a base teórica deste trabalho, a saber: *Database as a Service* e Consistência de Dados.

2.1. *Database as a Service*

O termo “nuvem” é uma alusão para um conjunto de servidores externos que hospedam um serviço. Quando esse serviço é um banco de dados, o modelo é conhecido como Banco de Dados como Serviço, do inglês *Database as a Service* (DBaaS).

O Banco de Dados como Serviço é uma área de interesse recente em Tecnologia da Informação. Uma das primeiras iniciativas para disponibilização de um banco de dados nesse ambiente surgiu em 2009. Nesse ano, a Amazon.com lançou o SGBD MySQL no âmbito da nuvem[10]. Desde então, a ideia vem evoluindo, inclusive com o surgimento de SGBD específicos para a nuvem, uma vez que os sistemas tradicionais se mostraram pouco eficientes para tal fim. Dentre os nativos para a nuvem, podem-se citar, de acordo com Sousa et al. (2010): Dynamo[11], Voldemort[12], BigTable[13], entre outros.

Um dos problemas encontrados nos SGBD tradicionais foi “*a falta de suporte para particionamento dinâmico eficiente de dados, o que limitou a escalabilidade e a utilização de recursos*” (ELMASRI e NAVATHE, 2011). O particionamento dinâmico refere-se à capacidade do sistema de realizar essa tarefa sem a interferência de um DBA. Esse aspecto é essencial ao DBaaS, uma vez que tais sistemas devem lidar com adição de *hardware* e balanceamento de carga em tempo de execução como faz, por exemplo, o SQL Azure[14].

[10] http://aws.typepad.com/aws/2009/10/introducing-rds-the-amazon-relational-database-service-.html
[11] http://aws.amazon.com/pt/dynamodb/
[12] http://www.project-voldemort.com/voldemort/
[13] http://research.google.com/archive/bigtable.html
[14] Conforme documentação disponível em: http://msdn.microsoft.com/pt-br/library/windowsazure/ee336241.aspx. Acesso: 21/08/2013.

Nas seções seguintes, serão abordadas características do DBaaS, suas vantagens e outros pontos relevantes do paradigma, de modo que tal base teórica facilite a compreensão da aplicação prática deste trabalho.

2.1.1. Conceituação

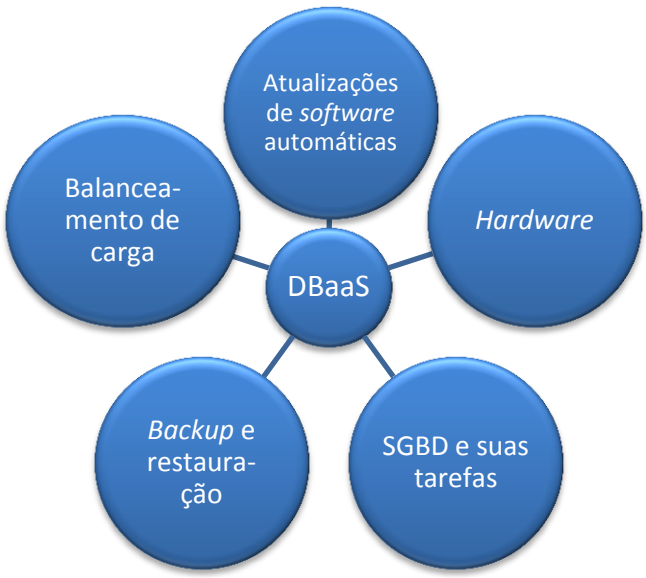
Diversos autores publicaram suas considerações sobre Banco de Dados como Serviço (HACIGÜMÜŞ et al., 2002b; AGRAWAL et al., 2009; CURINO et al., 2011). Não há, porém, consenso quanto à definição do paradigma. Serão relacionadas, a seguir, algumas definições.

De acordo com a Oracle (2011), DBaaS *"É uma abordagem arquitetural e operacional que permite aos provedores de TI entregar funcionalidades de banco de dados como um serviço para um ou mais consumidores*". Isso significa dizer que no modelo DBaaS, o serviço oferecido pelo provedor é formado por funcionalidades cotidianas dos SGBD, a saber: *backup* e recuperação de dados, balanceamento de carga, gerenciamento de memória, entre outros.

Para ALzain e Pardede (2011), DBaaS "*É um novo modelo para gerenciamento de dados onde o provedor de serviço oferece aos clientes funcionalidades de gerenciamento de software bem como o uso de hardware dispendioso*". Essa afirmação ressalta outro aspecto do DBaaS: a responsabilidade sobre o BD. A instalação do Sistema de Gerenciamento de Banco de Dados e a sua administração passam a ser responsabilidade do provedor. Dessa forma, o usuário não precisa mais preocupar-se com a compra de *hardware* e *software* custosos, nem lidar com atualizações de *software* ou mesmo contratar profissionais para tarefas administrativas e de manutenção (SAKR et al., 2011).

A Figura 2.1 enfatiza os serviços disponíveis em um DBaaS. Alguns exemplos de tarefas do SGBD, conforme mostra a figura, são: criar, acessar e manipular tabelas, *views*, índices, procedimentos, *triggers* e funções; além da capacidade de executar consultas complexas (ZHAO et al., 2012b).

Figura 2.1 – Algumas tarefas oferecidas por um DBaaS.



Fonte: A Autora.

As definições listadas mostram o propósito central do DBaaS, que é o provimento de um sistema de banco de dados com todas suas funcionalidades na nuvem. Será explanado, nas seções seguintes deste capítulo, o modo como tal objetivo é alcançado, por meio da exposição da arquitetura do modelo, bem como outras características relevantes.

2.1.2. Características

Por fazer parte do paradigma da Computação em Nuvem, o DBaaS inclui características análogas, como a escalabilidade, disponibilidade, baixo custo de investimento, entre outras.

A seguir, essas e outras características serão consideradas sob a óptica dos sistemas de bancos de dados em nuvem, de acordo com o que foi publicado em Cooper et al. (2009), da empresa Yahoo! Inc.:

- ***Escalabilidade*** - deve ser possível dar suporte a grandes bancos de dados, com altas taxas de requisição, e com uma latência muito baixa. O sistema deve ser capaz de escalar para assumir novos inquilinos ou lidar com o crescimento do número de inquilinos sem muito esforço, além de ser capaz de adicionar mais *hardware* facilmente. Em particular, o sistema deve redistribuir os dados automaticamente a fim de tirar vantagem do novo *hardware.*
  Deve ser possível também deslocar cargas automaticamente entre servidores (réplicas), isto é, prover balanceamento de carga (SAKR e LIU, 2012);

- ***Disponibilidade*** - A nuvem deve estar sempre em funcionamento. Embora possa haver falhas no servidor ou na rede, e até mesmo o *datacenter* como um todo possa ficar *offline*, os serviços na nuvem devem continuar disponíveis; e

- ***Multi-inquilinos*** - De acordo com Yu et al. (2012), um dos aspectos mais importantes do DBaaS é o multi-inquilino, no qual um grande número de bancos de dados com diferentes cargas de trabalho são alocados em um ambiente e compartilham recursos. Deve ser possível dar suporte a muitas aplicações (inquilinos) na mesma estrutura de *hardware* e *software*. Esses inquilinos devem ser capazes de compartilhar informações, mas devem ter seus desempenhos isolados um do outro. Por exemplo, um dia de grande tráfego no Yahoo! Mail não deve resultar em um aumento no tempo de resposta para os usuários do Yahoo! Messenger. Além disso, a adição de um novo inquilino deve exigir pouco ou nenhum esforço extra, além de assegurar que a capacidade do sistema provisionada para a nova carga seja suficiente.

Ainda no que concerne aos bancos de dados em nuvem, pode-se citar as seguintes características:

- ***Custo*** - No modelo de Computação em Nuvem, o pagamento do serviço se dá de acordo com o uso (SAKR e LIU, 2012). No modelo DBaaS não é diferente. Dessa forma, o preço do fornecedor será influenciado por aspectos como quantidade de dados armazenados, quantidade de memória utilizada, quantidade de instâncias do banco de dados, transferência de dados, entre outros;

- ***Acesso a partir de Qualquer Lugar*** - Uma vez que os recursos estão disponíveis através da rede (STIPIC e BRONZIN, 2012), os dados podem ser acessados a partir de qualquer lugar, não sendo necessário, por exemplo, estar conectado ao ambiente local da organização; e

- ***Medição de Serviço*** - Sistemas em nuvem devem controlar e otimizar automaticamente o uso de recursos, aproveitando-se dessa capacidade para medir em algum nível de abstração apropriado cada tipo de serviço. O uso de recursos pode ser monitorado, controlado e reportado, dando transparência tanto para o provedor quanto para o consumidor do serviço utilizado (STIPIC e BRONZIN, 2012). No caso dos bancos de dados em nuvem, todos os aspectos que influenciam na precificação dos serviços devem ser medidos de modo claro, como o espaço em disco alocado, quantidade de dados transferidos, número de instâncias utilizadas, entre outros.

Para que seja possível a disponibilização de tais serviços, o DBaaS possui uma arquitetura distribuída, a qual será abordada na seção seguinte.

2.1.3. Aspectos Arquiteturais do DBaaS

É importante observar que a arquitetura do DBaaS é sempre distribuída. E, assim como na Computação em Nuvem, o *hardware* utilizado para armazenamento de dados pode ser composto por até centenas de computadores com baixo poder de processamento e cujos recursos são compartilhados entre os clientes (XIONG et al., 2011).

Uma abordagem bastante utilizada no modelo em nuvem é a replicação, isto é, a cópia dos dados e armazenamento dos mesmos em mais de um local. De acordo com Elmasri e Navathe (2011), há três estratégias para replicação dos dados: replicação total, replicação parcial e ausência de replicação.

No primeiro caso, todo o banco de dados é replicado em diversos locais. No segundo, alguns fragmentos do banco de dados são replicados e outros não. Já na abordagem ausente de replicação, cada fragmento é armazenado em exatamente uma dada localização e todos os fragmentos são disjuntos.

A disponibilidade do DBaaS é alcançada devido à replicação dos dados. De acordo com Tanembaum e Steen (2006), se um sistema for replicado, é possível que ele continue a funcionar, mesmo após uma réplica falhar, simplesmente comutando (requisições) de uma réplica para outra.

A replicação também é útil quando um sistema precisa de escalabilidade. Ainda de acordo com Tanembaum e Steen (2006), há duas formas de escalar: em número e em área geográfica.

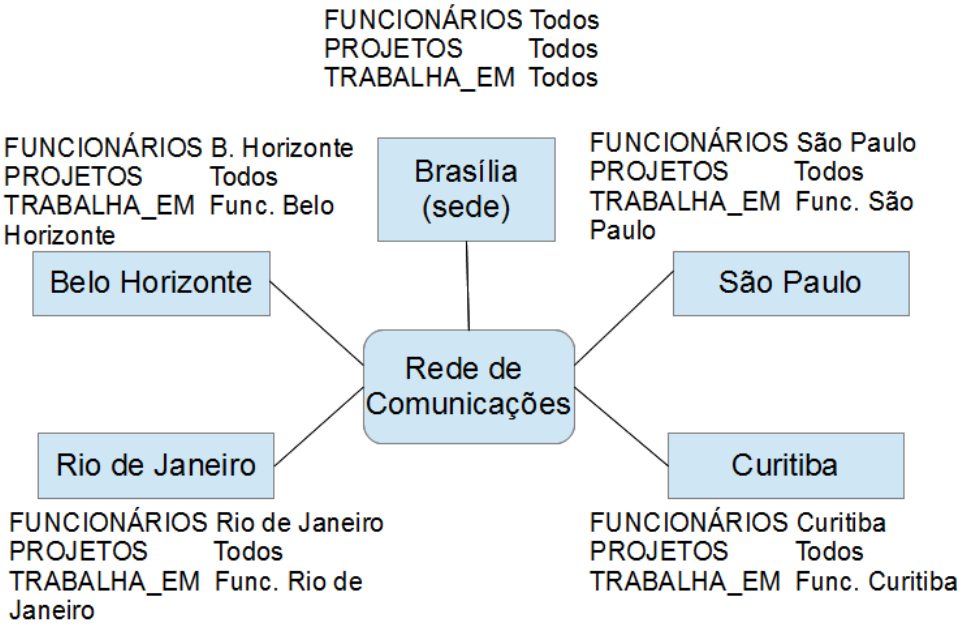
O primeiro caso ocorre, por exemplo, quando há um aumento no número de processos que precisam acessar os dados que são gerenciados por um servidor único. Nesse caso, o desempenho é aumentado replicando o servidor e subsequentemente dividindo o trabalho. Deste modo, é possível lidar com um número maior de requisições, armazenar maior quantidade de dados, além de alcançar o balanceamento de carga.

A replicação por área geográfica, por sua vez, tem por ideia central localizar os dados de acordo com a proximidade do processo que os utiliza. Nos sistemas de bancos de dados em nuvem, essa estratégia afeta diretamente o *throughput* – longas distâncias implicam em *throughput* baixo, conforme mostrado por Zhao et al. (2012b).

A Figura 2.2 mostra um exemplo de replicação de dados por área geográfica, no qual as cópias dos mesmos objetos de dados podem ser armazenadas em vários locais para melhor disponibilidade, desempenho e confiabilidade. Neste caso, o banco de dados de Brasília (sede) possui os dados

de todos os funcionários da empresa e cada filial possui os dados dos funcionários que atuam na cidade.

Figura 2.2 – Replicação de dados por área geográfica.



Fonte: Adaptado de Elmasri e Navathe, 2011.

A possibilidade de replicação dos dados em áreas significativamente distantes é outra característica essencial dos sistemas de banco de dados em nuvem (bem como o particionamento dinâmico de dados). Primeiramente, essa alternativa era viável apenas para grandes empresas, devido ao alto custo de se manter servidores físicos em diferentes regiões (Zhao et al., 2012).

Um dos aspectos importantes da replicação de dados é o modo como a disseminação das atualizações é realizada entre as réplicas. Diferentes abordagens podem ser utilizadas, as quais afetam diretamente a consistência dos dados armazenados. A replicação e sua relação com a consistência será abordada na Seção 2.2 deste trabalho.

Outra técnica de distribuição presente no DBaaS é o particionamento de dados. De acordo com Rahimi e Haug (2010), particionamento "*quebra uma tabela em dois ou mais pedaços chamados fragmentos ou partições e permite o*

armazenamento desses pedaços em diferentes locais". Note-se que, diferentemente da replicação, não ocorre cópia dos dados e sim a divisão dos mesmos, de acordo com um critério previamente definido. Em geral, o particionamento de dados é realizado manualmente, sob responsabilidade do DBA. Na nuvem, o particionamento é dinâmico, realizado em tempo real.

Por fim, o compartilhamento de recursos entre os clientes resulta em maximização de seu uso. Isso porque, uma vez que um cliente deixa de utilizar um determinado recurso, outro cliente poderá alocá-lo, mantendo as capacidades dos servidores ocupadas por mais tempo.

Os aspectos arquiteturais citados são responsáveis por diversos benefícios do DBaaS. Adicionalmente, tais aspectos específicos do modelo em nuvem podem trazer algumas dificuldades, as quais serão discutidas nas seções seguintes, que apresentam as vantagens e desvantagens do modelo de sistemas de banco de dados em nuvem.

2.1.4. Vantagens

A utilização da abordagem de Banco de Dados como Serviço possui diversas vantagens para o contratante, as quais a justificam. Pode-se citar, entre outras:

- ***Custo up-front*** - Refere-se à soma dos custos necessários para dar início a um projeto, que, neste caso, engloba toda a infraestrutura para dispor de um banco de dados. Esse custo, em geral, envolve a aquisição de um ou mais servidores com alta capacidade de processamento e armazenamento, contratação de profissionais para instalação e gerenciamento do SGBD, além de aquisição de licenças do *software* necessário. Tudo isto implica em um alto custo de investimento inicial. No DBaaS, o custo *up-front* é minimizado, uma vez que toda a arquitetura e gerenciamento do serviço são disponibilizados pelo provedor;

- ***Custos mais baixos*** - Segundo Curino et al. (2011), devido à economia de escala, os custos de *hardware* e de energia incorridos pelos usuários tendem a ser muito menores quando eles estão pagando por uma parcela de um serviço ao invés de arcar com tudo por conta própria. Adicionalmente, ao centralizar e automatizar muitas tarefas de gerenciamento de banco de dados, um DBaaS pode reduzir substancialmente os custos operacionais;

- ***Complexidade técnica reduzida*** - Uma vez que as funções administrativas e de manutenção são de responsabilidade do provedor (HACIGÜMÜS et al., 2002), a intensa dificuldade relativa a essas tarefas é suprimida; e

- ***Time-to-market reduzido*** - *Time-to-market* diz respeito ao tempo que um produto demora em chegar ao mercado. A diminuição desse tempo é uma vantagem da abordagem DBaaS (SAKR e LIU, 2012). O motivo disto é que o tempo gasto em compra de equipamentos, instalação de servidores e planejamento da arquitetura de *hardware* é eliminado. No ambiente desenvolvido pelo provedor, todos esses passos já foram realizados, sendo necessário planejar apenas o esquema do banco de dados e afins.

Estudos de caso divulgados no *site* da Amazon, provedora de sistemas de banco de dados em nuvem, mostram que diversas empresas têm se beneficiado do paradigma de DBaaS, como é o caso da Sega[15], ThoughtWorks[16] e até mesmo a própria Amazon[17]. Tais adoções mostram que o DBaaS vem crescendo e ganhando força com o passar dos anos, tornando-se estratégia de armazenamento de várias empresas, além de ser foco de estudo de diferentes trabalhos acadêmicos, conforme indicam as referências utilizadas neste trabalho.

[15] http://aws.amazon.com/pt/solutions/case-studies/sega. Acesso: 25/10/2012.
[16] http://aws.amazon.com/pt/solutions/case-studies/mccann-erickson-thoughtworks. Acesso: 25/10/2012.
[17] http://aws.amazon.com/pt/solutions/case-studies/amazon-cxa. Acesso: 25/10/2012.

2.1.5. Desvantagens

A adoção do modelo DBaaS, é patente, também tem suas dificuldades. Algumas delas são:

- ***Segurança / Privacidade*** - Uma das principais preocupações das empresas no que se refere à adoção dos bancos de dados em nuvem é a segurança dos dados armazenados. Conforme argumentam Weitzner et al. (2008), controle de acesso e criptografia, somente, não são capazes de proteger a privacidade desses dados. Diversos trabalhos que tratam desse aspecto do DBaaS têm sido publicados, como os de Cheung (2011), Pavlou e Snodgrass (2012), Arasu et al. (2013) e Lenkala et al. (2013), no entanto, ainda não há consenso quanto à solução definitiva para o problema;

- ***Localização dos dados*** - Cada provedor adota uma estratégia de localização dos servidores que hospedam os BD, que pode ser a proximidade com o país do usuário, por exemplo, a fim de diminuir o tempo de resposta das requisições. Em cada caso, no entanto, é preciso cautela em relação às leis locais, uma vez que os dados armazenados nesses servidores estão sujeitos a tais normas. Em alguns países, os dados podem até mesmo ser acessados sem prévio conhecimento dos usuários dos serviços terceirizados (ABADI, 2009); e

- ***Migração de nuvem*** - Apesar de ter os custos *up-front* reduzidos no caso de novas aplicações, esta pode não ser a situação para sistemas legados que são migrados para a nuvem. Isso porque tais sistemas possuem, em geral, vários *gigabytes* de dados já armazenados, o que aumenta o tempo de transferência do servidor local à nuvem, além de ter um custo associado – o qual cresce na medida em que a quantidade de dados aumenta. Adicionalmente, Shirazi et al. (2012) argumentam que a dificuldade técnica pode

> fazer da migração de um sistema de uma nuvem a outra um projeto sem possibilidade de execução. Isso se dá em razão das diferentes arquiteturas de armazenamento utilizadas pelos provedores dos serviços.

No artigo intitulado "*Nuvem Relacional: um Banco de Dados como Serviço para a Nuvem*", Curino et al. (2011) afirmam que a privacidade dos dados é um dos pontos fundamentais para que a terceirização do serviço de banco de dados se torne popular. É preciso que o provedor dê garantias aos usuários de que, além de segurança, pode oferecer qualidade do serviço, tempo de resposta, disponibilidade, entre outras garantias.

Diante das dificuldades apresentadas anteriormente, faz-se necessário um conjunto de medidas que venham a minimizar tais problemas, provendo uma maior segurança aos usuários dos serviços hospedados em nuvem. Para este fim, os Contratos de Nível de Serviço, SLA, podem ser adotados.

2.1.6. *Service Level Agreements* (SLA)

Segundo SAKR e LIU (2012), os SLA "*representam o contrato que capta as garantias acordadas entre um provedor de serviços e seus clientes*". Para Weis e Alves-Foss (2011), os SLA "*são o único documento legal entre o provedor e o cliente*".

O contrato é a alternativa para que o cliente tenha garantias de qualidade do serviço adquirido, uma vez que certas configurações de baixo nível somente podem ser realizadas pelo servidor.

O trabalho de Suleiman et al. (2012) divide os contratos de nível de serviço em duas categorias, a saber: SLA de Infraestrutura de Nuvem e SLA de Aplicações baseadas em Nuvem.

A primeira categoria diz respeito ao acordo entre o provedor de serviços e seus clientes, os quais hospedam serviços em nuvem. Engloba especificações como desempenho do servidor, velocidade da rede, disponibilidade de recursos e capacidade de armazenamento, entre outras.

Já a segunda se refere ao relacionamento entre clientes da nuvem e os usuários finais das aplicações, uma vez que os clientes também devem garantir a qualidade dos seus serviços. Envolve questões como tempo de resposta, disponibilidade, segurança e tempo médio de espera, entre outras.

Ademais, é preciso que esteja claro ao cliente quais as medidas que deverão ser tomadas pelo provedor do serviço com relação à recuperação de desastres, destruição de dados, acesso de usuários privilegiados, entre outras dificuldades que possam vir a ocorrer durante a prestação do serviço (WEIS e ALVES-FOSS, 2011). Por fim, caso o serviço não atinja os níveis acordados, são definidas penalidades para os provedores. Tal punição pode ser, por exemplo, redução no preço pago pelo serviço.

Atualmente, os SLA disponíveis não atendem às necessidades dos usuários. Muitos provedores garantem, em contrato, apenas a disponibilidade de seus serviços (SAKR e LIU, 2012), ignorando outras competências relevantes para dar segurança ao cliente na contratação do serviço, como o tempo de resposta às requisições, critérios de replicação dos dados, entre outros.

A especificação detalhada do contrato de nível de serviço pode ser um incentivo à adoção da nuvem por parte das empresas, particularmente aquelas cujos serviços são críticos e precisam de garantias. Deste modo, o avanço desse tema deve ser amplamente estimulado.

A seção seguinte descreve os sistemas do tipo OLAP e OLTP, os quais divergem, especialmente, pelo modo como são realizadas as atualizações. Essa característica particular tem influência direta no tratamento da consistência de dados nesses dois tipos de aplicações, conforme será mostrado a seguir.

2.1.7. Tipos de Sistemas (OLTP / OLAP)

De maneira ampla, pode-se classificar as aplicações em dois tipos: (1) sistemas de dados transacionais – *Online Transaction Processing* (OLTP); e (2) sistemas de dados analíticos - *Online Analytic Processing* (OLAP).

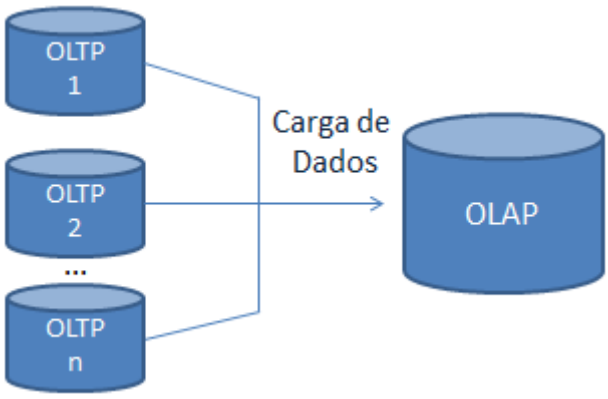
Segundo Bog et al. (2009), sistemas de dados transacionais:

> São aqueles que permitem as operações diárias de negócios de uma empresa. Todos os pedidos recebidos, eventos e informações de clientes, fornecedores e outros parceiros de negócios, bem como processos de produção, são registrados, monitorados e processados.

Nesses sistemas, os usuários estão constantemente realizando alterações (inserções, atualizações, remoções) nos dados. Já os sistemas de dados analíticos não são atualizados pelos usuários, mas sim por cargas de dados. Ainda segundo Bog et al. (2009), são coletadas informações estratégicas geradas por meio dos sistemas OLTP. Tais informações serão úteis na tomada de decisão.

A Figura 2.3, a seguir, mostra esse processo: o conjunto de dados OLAP é atualizado mediante cargas periódicas realizadas com dados armazenados em sistemas de dados transacionais (OLTP).

Figura 2.3 - Atualização do sistema de dados analíticos (OLAP).



Fonte: A Autora.

Dentre esses modelos, a literatura tem destacado o OLAP como o mais bem adaptado ao paradigma de DBaaS. Com relação a isso, Abadi (2009) conclui:

> Devido à necessidade cada vez maior de análises sobre muito mais dados no mundo corporativo atual, somado à combinação arquitetural de desenvolvimento disponível atualmente, conclui-se que aplicações de gerenciamento de dados analíticos são mais bem adequadas para a implantação na nuvem que aplicações de gerenciamento de dados transacionais.

Além dos motivos citados por Abadi (2009), a consistência dos dados é um fator que também influencia essa avaliação. Nos sistemas OLAP, há um maior número de leituras em relação às escritas, que não ocorrem concorrentemente.

Essas características facilitam o tratamento da consistência. Deste modo, o modelo de consistência eventual, no qual os SGBD em nuvem atuais têm se apoiado (CHIHOUB et al., 2012), supre as necessidades deste tipo de aplicação.

Essas situações, no entanto, não inviabilizam a utilização da nuvem para disponibilização de sistemas OLTP. Soluções que garantam a consistência dos dados, foco desta pesquisa, beneficiam sistemas desse tipo. Isso porque as consultas executadas em um ambiente consistente recuperam o dado mais recentemente armazenado, característica essencial para o correto funcionamento dos sistemas de dados transacionais, que lidam com constantes atualizações e acesso concorrente dos usuários. A garantia da consistência permite, portanto, a utilização dos sistemas OLTP na nuvem.

A seguir, serão abordados os conceitos relacionados à consistência de dados, descrevendo-se diversos modelos de consistência existentes, com vistas a fundamentar o modelo proposto no presente trabalho.

2.2. Consistência de Dados

A Consistência de Dados é uma característica bastante relevante dos Sistemas de Gerenciamento de Banco de Dados tradicionais. Uma transação, que pode ser constituída por uma sequência de operações de leitura e escrita na base de dados (ÖZSU e VALDURIEZ, 2011), possui as seguintes características desejáveis, de acordo com Elmasri e Navathe (2011):

- ***Atomicidade*** - Uma transação deve ser executada por completo ou não ser realizada;
- ***Preservação da Consistência*** - Uma transação deve levar o banco de um estado consistente a outro;
- ***Isolamento*** - Embora muitas transações sejam executadas simultaneamente em um SGBD, cada transação deve ser executada como se fosse única, sem interferências; e

- ***Durabilidade*** - Todas as mudanças realizadas no banco de dados devem ser persistidas e não devem ser perdidas em razão de falhas.

Essas propriedades, conhecidas como ACID, foram amplamente discutidas em sistemas de banco de dados tradicionais e há diferentes estratégias para alcançá-las completamente. Nos sistemas de bancos de dados em nuvem, no entanto, suas características arquiteturais, destacadas anteriormente, tornam a obtenção desses atributos um processo não trivial.

Uma das dificuldades na manutenção das garantias ACID é a replicação (ABADI, 2009). De acordo com Özsu e Valduriez (2011), *"Um banco de dados replicado está em um estado mutuamente consistente se todas as réplicas de cada um dos seus elementos de dados têm valores idênticos*". Para que isso ocorra, faz-se necessário a utilização de estratégias de atualização e propagação de dados, conforme observam Tanenbaum e Steen (2006): "*quando uma cópia é atualizada, é preciso garantir que as outras cópias também são atualizadas*".

As questões que afetam a consistência dos dados serão explanadas nesta seção. Vale ressaltar que esse tópico será tratado de modo a evidenciar a replicação de dados, devido ao seu emprego constante em sistemas de banco de dados em nuvem. Deste modo, os dois temas serão tratados conjuntamente, a fim de facilitar a compreensão de ambos os tópicos.

2.2.1. Teorema CAP

Conforme provado formalmente por Gilbert e Lynch (2002), não é possível atingir em um sistema distribuído, e ao mesmo tempo, as desejáveis características de consistência, disponibilidade e tolerância à partição (em caso de falha).

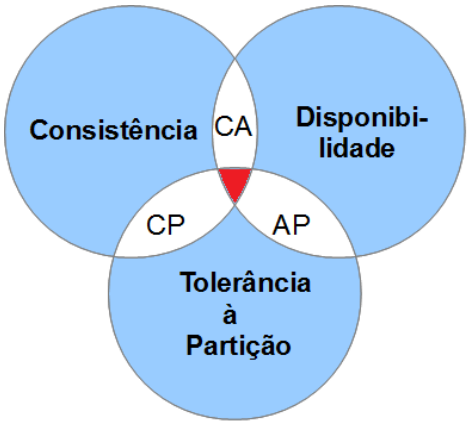
Tal proposição foi concebida por Eric Brewer em 2000 e ficou conhecida como Teorema CAP (***C****onsistency,* ***A****vailability,* ***P****artition Tolerance)* – em português: Consistência, Disponibilidade e Tolerância à Partição. Os sistemas de banco de dados em nuvem, portanto, se encaixam nesse perfil.

De acordo com Zhao et al. (2012), consistência significa ter todos os registros iguais em todas as réplicas, conforme explanado na seção anterior. Já a

disponibilidade indica que todas as réplicas podem aceitar atualizações ou inserções. Por fim, tolerância à partição é manter o sistema em funcionamento mesmo que réplicas distribuídas tenham problema de comunicação umas com as outras.

Simplificadamente, pode-se afirmar que no máximo duas das três características CAP podem ocorrer ao mesmo tempo em caso de falha. Por exemplo, um sistema pode garantir disponibilidade e tolerância à partição, e não garantir consistência. A Figura 2.4, mostra uma representação visual do Teorema CAP.

Figura 2.4 - Teorema CAP: apenas duas características podem ocorrer plenamente em um ambiente distribuído.



Fonte: Adaptado de CouchDB: The Definitive Guide – Eventual Consistency[18].

O próprio Brewer (2012), no entanto, defende que a consistência ou disponibilidade não sejam "cegamente" sacrificadas quando houver partições de rede. Segundo o referido autor, deve-se, ao contrário, "*otimizar as propriedades por meio de uma gestão cuidadosa das invariantes*". Ou seja, é preciso gerenciar adequadamente as divergências entre os dados de modo a combinar corretamente a necessidade de consistência dos dados das aplicações.

Uma variante do teorema CAP é o PACELC, que será explanado na seção seguinte.

[18] Disponível em: <http://guide.couchdb.org/editions/1/en/consistency.html>. Acesso: 17/05/2013.

2.2.2. PACELC

O PACELC apresenta uma visão diferente do Teorema CAP e foi proposto por Daniel Abadi (2012). Segundo o autor, o PACELC é uma descrição mais completa das potenciais vantagens e desvantagens de consistência para um sistema de banco de dados distribuído. Sua descrição é a seguinte:

Se há uma partição (P), como o sistema resolve o conflito entre disponibilidade e consistência (A e C); do contrário (E), quando o sistema está executando normalmente na ausência de partições, como ele resolve o conflito entre latência (L) e consistência (C)?

Isto é, em caso de partição na rede, o sistema precisa escolher se fornecerá ao usuário disponibilidade ou consistência. Caso não haja partições, o conflito a ser resolvido é entre latência e consistência.

O PACELC é, portanto, uma forma diferente de visualizar o problema da consistência, disponibilidade e tolerância à partição na nuvem, ratificando que o relaxamento da consistência não é a única alternativa para os sistemas de banco de dados em nuvem.

A seção seguinte explica os dois tipos de consistência, forte e fraca, e suas diferenças.

2.2.3. Tipos de Consistência

O modo como a consistência dos dados é tratada pode ser classificado em dois grandes grupos: Consistência Forte e Consistência Fraca. Há, no entanto, diferentes nuances dentro desses conjuntos, especialmente no segundo. Esta seção aborda ambos os grupos e as diferenças entre eles.

Consistência Forte

Um SGBD que oferece garantia de consistência está lidando com o tipo de Consistência Forte. Nesse modelo, o usuário tem a certeza de que qualquer modificação (inserção, atualização ou remoção) realizada no banco de dados será vista por qualquer outro usuário que consultar esse mesmo dado.

De acordo com Ramakrishnan (2012), consistência forte "*significa que uma vez que uma solicitação de gravação retorna com êxito para o cliente, todas as leituras subsequentes do objeto - por qualquer cliente - verão o efeito da escrita, independentemente da replicação, falhas, partições, e assim por diante*".

É preciso observar, no entanto, que quanto maior o nível de garantia de consistência, maior será o custo necessário para aplicá-la (FETAI e SCHULDT, 2012).

Consistência Fraca

A Consistência Fraca, por sua vez, "*descreve qualquer alternativa que não garante a consistência forte para alterações em objetos individuais*" (RAMAKRISHNAN, 2012). Em outras palavras, não há garantias de que, após a escrita, os usuários que consultarem o dado atualizado receberão sua versão mais recente.

Uma das variações mais conhecidas desse modelo é a consistência eventual, que garante apenas que os dados serão tornados consistentes eventualmente (ISLAM et al, 2012). Nesse caso, a propagação dos dados é realizada de modo a assegurar que as atualizações chegarão a todas as outras réplicas, sem garantir, no entanto, o momento em que esse fato deverá ocorrer.

A consistência eventual possui diversas variações, podendo-se destacar, de acordo com Vogels (2009):

- ***Consistência Causal (Causal Consistency)*** - Se um processo A comunicou ao processo B que atualizou um item de dados, um acesso subsequente pelo processo B retornará o valor atualizado, e uma gravação é garantida para substituir a gravação anterior. O acesso pelo processo C que não tem nenhuma relação causal para processar A está sujeito às regras normais de consistência eventual;

- ***Leia-suas-escritas (Read-your-writes consistency)*** - Este é um modelo importante onde o processo A, depois de ter atualizado um item de dados,

sempre acessa o valor atualizado e nunca verá um valor mais antigo. Este é um caso especial do modelo consistência causal;

- ***Consistência de Sessão (Session consistency)*** – Esta é uma versão prática do modelo anterior, na qual um processo acessa o sistema de armazenamento no contexto de uma sessão. Enquanto existir a sessão, o sistema garante a consistência do tipo leia-suas-escritas. Se a sessão termina por causa de um determinado cenário de falha, uma nova sessão precisará ser criada e as garantias não se sobrepõem às sessões;

- ***Consistência de Leitura Monotônica (Monotonic read consistency)*** - Se um processo já recebeu um determinado valor para o objeto, os acessos subsequentes nunca retornarão quaisquer valores anteriores; e

- ***Consistência de Escrita Monotônica (Monotonic write consistency)*** - Neste caso, o sistema garante serializar as escritas pelo mesmo processo. Sistemas que não garantem esse nível de consistência são notoriamente difíceis de programar.

Além dos exemplos apresentados, há diversos outros em uso atualmente. Alguns deles serão descritos na Seção 2.2.5, na descrição de SGDB em nuvem disponíveis no mercado.

Para que o projetista possa escolher o modelo mais adequado, faz-se necessário conhecer o tipo de aplicação a ser desenvolvida. Diversas aplicações permitem que as consultas retornem dados desatualizados – especialmente as aplicações de Web 2.0 - permitindo, assim, o uso da consistência fraca. Entretanto, há o grupo de aplicações que exige a leitura de dados consistentes, necessitando, portanto de garantia de consistência forte.

Por fim, pode-se observar que uma mesma aplicação pode conter dados que necessitam de consistência forte e outros que podem utilizar-se da consistência fraca. Tais aplicações, híbridas, também serão tratadas no presente trabalho.

A seção seguinte apresenta diferentes abordagens de replicação e propagação de dados, explanados de modo a enfatizar o tratamento da consistência de dados.

2.2.4. Abordagens de Replicação e Propagação de Dados

A replicação de dados é um procedimento que consiste na criação de cópias dos dados e armazenamento das mesmas em lugares diferentes. De acordo com Sathya et al. (2006): "*A ideia geral da replicação é armazenar cópias de dados em diferentes locais de modo que os dados possam ser facilmente recuperados se uma cópia localizada em determinado lugar for perdida*". Uma vez que os benefícios desse procedimento já foram discutidos, serão tratadas, a partir de então, as abordagens utilizadas para esse fim.

A replicação de dados pode ser classificada em síncrona e assíncrona. De acordo com Rahimi e Haug (2010), no primeiro caso "*Uma transação pode acessar qualquer cópia do item de dados com a garantia de que o item de dados que está acessando tem o mesmo valor que todas as suas outras cópias*". Ou seja: na replicação síncrona há a garantia de consistência forte.

Ainda segundo Rahimi e Haug (2010), na replicação assíncrona "*Duas ou mais réplicas do mesmo item de dados podem ter valores diferentes, por vezes, e qualquer transação pode ver esses valores diferentes*". Assim, essa abordagem lida com a consistência fraca.

O que determina essa classificação é o modo como são realizadas as atualizações e a propagação dos dados às outras réplicas. Pode-se optar entre a atualização preguiçosa (*lazy)* ou ansiosa (*eager)* – a qual garante a consistência - que funcionam como descrito a seguir:

Atualização Ansiosa

Nessa abordagem, todas as atualizações são realizadas no contexto da transação. Consequentemente, quando a transação é confirmada, todas as réplicas possuem o mesmo valor (ÖZSU e VALDURIEZ, 2011).

A atualização ansiosa garante a serialização, isto é, a ordem de execução das operações das transações é sempre considerada correta quando transações concorrentes estão sendo executadas (ELMASRI e NAVATHE, 2011).

Uma das estratégias que pode ser utilizadas por esse modelo é o *commit* em duas fases (*two-phase commit* - 2PC) (ÖZSU e VALDURIEZ, 2011), que divide a execução do *commit* em duas etapas: na primeira, o coordenador envia um aviso de ‘preparar para *commit*’ a todos os servidores envolvidos na transação; na segunda fase, o coordenador pode enviar a mensagem ‘realizar o *commit*’ (caso tenha recebido todas as respostas positivas) ou abortar a transação.

A atualização ansiosa, entretanto, pode causar problemas de desempenho, uma vez que durante a execução da transação, o dado a ser atualizado fica bloqueado (RAHIMI e HAUG, 2010). Desse modo, é preciso que todas as réplicas sejam atualizadas para que outras transações possam acessá-los.

Ademais, de acordo com Özsu e Valduriez (2011), se uma das réplicas falhar, a transação não poderá ser finalizada. Desse modo, alternativas devem ser buscadas, como a comunicação em grupo, utilizada no modelo de consistência proposto neste trabalho.

Özsu e Valduriez (2011) explicam, ainda, que além da garantia de consistência, pode-se citar como vantagem da atualização ansiosa também a capacidade de atualizar as réplicas atomicamente, o que permite a utilização de protocolos conhecidos de recuperação de falhas.

Atualização Preguiçosa

Nesse modelo, a atualização é realizada em uma réplica e repassada às outras réplicas de modo assíncrono. Como Gao e Diao (2010) afirmam: “*algoritmos de propagação preguiçosa postam as atualizações para as réplicas por meio de transações independentes, após a confirmação da transação de atualização no local de origem”*. Deste modo, a atualização dos dados é um processo separado da propagação.

Uma das estratégias de atualização preguiçosa, utilizada por diferentes SGBD tradicionais, é o *store and forward* (armazenar e transmitir), no qual um local primário é atualizado inicialmente e repassa as atualizações enfileiradas às réplicas, que as aplica em lotes. Para preservar a consistência dos dados, o local primário deve determinar a ordem de serialização das transações (RAHIMI e HAUG, 2010). Outras estratégias, as quais não utilizam locais primários, também podem ser utilizadas. Para maiores detalhes recomenda-se consultar o trabalho dos autores supracitados.

Entre as vantagens desse modelo, pode-se citar a diminuição do tempo de resposta e a capacidade de lidar com um maior número de requisições (GAO e DIAO, 2010). Apesar disso, não há garantia de consistência mútua das réplicas.

O Quadro 2.1, mostra um comparativo entre as características dos dois modelos anteriormente discutidos.

Quadro 2.1 - Síntese das características dos modelos de atualização ansiosa e preguiçosa.

Tipo de atualização	Método	Tipo de consistência	Vantagem	Desvantagem
Ansiosa	Todas as atualizações realizadas no contexto da transação	Consistência forte	Permite a utilização de protocolos conhecidos de recuperação de falhas	Alto custo de implementação
Preguiçosa	A atualização é realizada em uma réplica e repassada às outras réplicas de modo assíncrono	Consistência fraca	Diminuição do tempo de resposta e capacidade de lidar com um maior número de requisições	Não é adequado para aplicações que necessitam de consistência mútua entre réplicas

Fonte: A Autora.

Na seção seguinte, há um estudo comparativo entre diferentes SGBD em nuvem disponíveis no mercado. São apresentados os principais aspectos relacionados à consistência de dados em cada um dos SGBD analisados.

2.2.5. SGBD em Nuvem e suas Características de Consistência

Desde a disponibilização do primeiro SGBD no ambiente em nuvem, diversos avanços já foram realizados. Inicialmente, os SGBD já consagrados

foram dispostos nesse ambiente, como é o caso do já citado MySQL, disponibilizado pela Amazon.com.

Outros sistemas de bancos de dados, porém, foram desenvolvidos no intuito de suprir as exigências do ambiente na nuvem. Esses SGBD, com características pensadas para operar de modo adequado na nuvem, são denominados nativos.

Esta seção tem por objetivo abordar diferentes SGBD utilizados no ambiente em nuvem. Serão discutidos tanto SGBD relacionais quanto não relacionais, a fim de verificar o tipo de tratamento dado à consistência dos dados e o modo como o usuário interfere no modelo. Desse modo, é possível ainda verificar a oferta real da nuvem com relação aos sistemas de banco de dados.

Os aspectos discutidos serão os seguintes: Ambiente no qual o SGBD foi desenvolvido (comercial ou acadêmico), a fim de compreender o tipo de tratamento que ambas as partes têm dado ao tema; tipo de garantia de consistência fornecida pelo sistema e funcionamento da mesma; e, por fim, se há a possibilidade de interferência do usuário na definição do tipo de consistência utilizado pelo SGBD.

Serão tratados, inicialmente, os SGBD não relacionais:

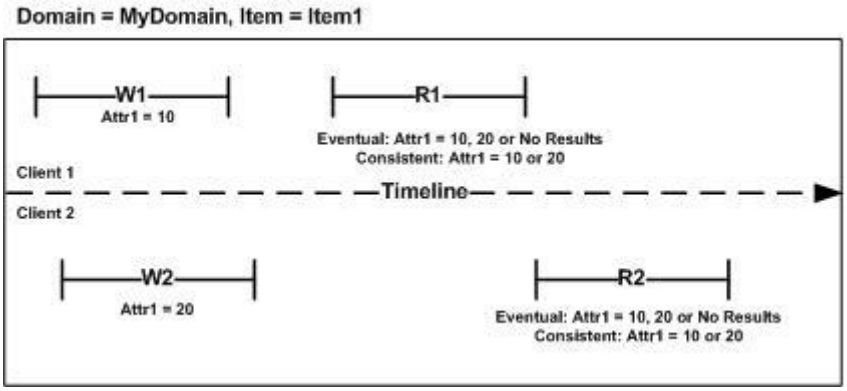
Amazon SimpleDB

O Amazon SimpleDB é um sistema de banco de dados não-relacional desenvolvido pela Amazon.com. Até Fevereiro de 2010, dava suporte apenas para leituras eventualmente consistentes. Entretanto, a partir de então, esse sistema de banco de dados passou a dar suporte a leituras consistentes. Segundo sua documentação oficial[19], isso significa dizer que ele retorna um resultado que reflete todas as gravações que receberam uma resposta bem sucedida antes da leitura. No entanto, a leitura pode não retornar a última gravação se houver sobreposições de leitura ou escrita, conforme exemplo abaixo, contido na documentação supracitada:

[19] http://aws.amazon.com/articles/Amazon-SimpleDB/3572

Cenário - Os clientes 1 e 2 recebem respostas bem-sucedidas das escritas W1 e W2, antes de enviar R1 e R2, respectivamente. Há uma sobreposição das escritas W1 e W2.

Figura 2.5 – SimpleDB: Leitura pode retornar valores inconsistentes com sobreposição de escritas



Fonte: Documentação SimpleDB. Disponível em: http://aws.amazon.com/articles/Amazon-SimpleDB/3572. Acesso: 02/06/2013.

Conforme mostrado na Figura 2.5, os resultados obtidos serão os seguintes:

- Se R1 ou R2 forem leituras consistentes, retornarão o mesmo resultado. No entanto, esse resultado pode ser Attr1 = 10 ou Attr1 = 20, em razão da sobreposição de escritas W1 e W2. Observe-se que, mesmo W2 tendo sido concluída após W1, o resultado retornado poderá ignorar essa escrita; e

- Caso R1 e R2 sejam leituras eventualmente consistentes, os resultados retornados podem ser Attr1 = 10, Attr1 = 20 ou sem resultado. Ademais, R1 e R2 podem retornar resultados diferentes.

Bigtable

O Bigtable é um sistema de armazenamento para gerenciar dados estruturados que são projetados para escalar até tamanhos muito grandes (CHANG et al., 2006). Diversos serviços do Google Inc. armazenam dados no Bigtable.

No tocante à consistência, o Bigtable utiliza o método *single-row transaction*. Deste modo, de acordo com Ramanathan et al. (2011), o sistema pode garantir a consistência dos dados apenas para transações em um único item de dados.

Em outras palavras, a garantia de consistência é dada apenas para dados armazenados na mesma linha e acessados em uma mesma transação. Desse modo, se dois campos de uma mesma linha são atualizados e, por alguma razão, uma das operações não for completada, nenhum dos valores será escrito.

Cassandra

O sistema de banco de dados Apache Cassandra é, por padrão, eventualmente consistente. No entanto, ele permite que o usuário escolha o nível de consistência desejado para leitura e escrita, que pode ser, entre outros: ALL, ONE, QUORUM.

De acordo com a sua documentação[20], o processo funciona de acordo com a consistência do tipo *Quorum*: Se W + R > N, será alcançada a consistência. Sendo R o número de nós a serem bloqueados na leitura, W o número de nós a serem bloqueados na escrita e N o fator de replicação, ou seja, o número de réplicas que armazenam a chave (se o fator de replicação é 1, cada chave é armazenada em apenas um nó).

O nível de consistência ONE significa que R ou W é igual a 1. O nível de consistência QUORUM significa que R ou W é igual ao arredondamento de (N+1)/2. O nível de consistência ALL significa que R ou W é igual a N.

Então, para escrever com o nível de consistência ONE e obter consistência, é necessário selecionar o nível de consistência de escrita ALL (que significa que os valores só serão retornados à consulta se todas - *all* - as réplicas possuírem o mesmo valor).

[20] http://wiki.apache.org/cassandra

O sistema poderá retornar um valor incorreto caso haja falha em uma das réplicas, uma vez que no momento em que ela se recuperar da falha, o valor armazenado na mesma pode estar incorreto e mesmo assim aceitar consultas.

Vale ressaltar, ainda, que, a depender da escolha do usuário, pode haver um aumento do número de transações rejeitadas. Por exemplo, se o nível de consistência de escrita escolhida for ONE e a réplica falhar, a escrita será rejeitada. A opção ONE é simplesmente a ausência de replicação.

G-Store

O G-Store é um sistema de armazenamento de dados do tipo chave-valor (do inglês, *key-value*). De acordo com Connor et al. (2011), tais sistemas são estruturas de dados em que há uma chave primária e um objeto, os quais só podem ser acessados por meio da chave.

De acordo com Das et al. (2010), o armazenamento chave-valor provê garantias de consistência apenas para acessos de chave única, ou seja, para um mesmo registro.

O G-Store, por sua vez, utiliza o conceito de grupos de entidades. Ainda de acordo com Das et al. (2010), nesse modelo, as aplicações podem selecionar membros para um grupo a partir de qualquer conjunto de chaves em tempo de execução permitindo, então, acesso consistente ao conjunto de chaves escolhido. O processo ocorre da seguinte maneira: O protocolo de Agrupamento de Chave transfere o acesso exclusivo de leitura e escrita de todas as chaves de um grupo para um único nó. Cada grupo de chaves possui um líder associado e o nó que possui o líder do grupo é dado como proprietário do grupo. Então, todos os acessos de leitura e escrita para os membros de um grupo são fornecidos pelo líder, que pode garantir acesso consistente para as chaves sem a necessidade de utilização da sincronização distribuída, uma vez que a propriedade de todas as chaves de um grupo se encontra em um mesmo nó.

Além dos sistemas apresentados acima, convém discutir o tratamento para a consistência dado por SGBD relacionais dispostos na nuvem, uma vez que a solução apresentada nesta dissertação tem como foco este modelo de dados. Este trabalho será realizado a seguir:

AmazonRDS[21]

O AmazonRDS é o SGBD relacional disponibilizado pela Amazon.com. Este sistema de banco de dados é nativamente centralizado e permite duas opções de replicação: utilização de réplicas de leitura ou a utilização de réplica de espera, que é acionada apenas em caso de falha da réplica principal.

O AmazonRDS utiliza a replicação assíncrona nativa do MySQL e, no caso das réplicas de leitura, não garante a consistência dos dados, uma vez que pode haver inconsistência ou atraso entre uma réplica e sua instância.

Google Cloud SQL[22]

No Google Cloud SQL, os dados são armazenados sincronamente em múltiplos *datacenters.* Para garantir a consistência de dados este SGBD utiliza o conceito de grupos de entidades.

Um grupo de entidades é um espaço estruturado hierárquico, semelhante a uma estrutura de diretórios. Quando uma entidade é criada, seu pai pode ser designado. Desse modo, a consistência é garantida para o grupo de entidades e uma consulta do tipo *ancestor* não retorna o resultado até que todos os dados do grupo estejam atualizados.

Windows Azure[23]

Neste SGBD, os dados são *triple-replicated*, isto é, todos os dados são, por padrão, replicados em três localizações diferentes (em uma réplica primária e duas secundárias). Todas as consultas são realizadas no nó *master* e, portanto, não há preocupação com a consistência de dados distribuídos. A réplica *secundária* pode tornar-se *master*, em caso de falha da principal.

[21] http://aws.amazon.com/pt/rds/faqs/. Acesso: 09/11/2013.
[22] https://developers.google.com/appengine/docs/python/datastore/structuring_for_strong_consistency. Acesso: 09/11/2013.
[23] http://blogs.msdn.com/b/windowsazure/archive/2012/07/30/fault-tolerance-in-windows-azure-sql-database.aspx. Acesso: 09/11/2013.

Yahoo! PNUTS

O PNUTS (*Platform for Nimble Universal Table Storage*) foi desenvolvido para dar suporte às aplicações *web* do Yahoo! Este SGBD utiliza um modelo relacional próprio, chamado pelos autores de relacional simplificado (COOPER et al., 2008). Para o tratamento da consistência, utiliza o modelo de "*timeline consistency*", no qual todas as réplicas de um dado registro aplicam as alterações neste registro na mesma ordem (COOPER et al., 2008).

De acordo com Ramakrishnan (2012), o seguinte cenário ocorre:

> Em um determinado momento, cada registro tem exatamente uma cópia mestre. As atualizações são aplicadas neste mestre e, em seguida, propagadas para outras cópias, garantindo assim uma ordem única de todas as atualizações a um registro.

Desse modo "*um objeto e suas réplicas não precisam ser mantidos de forma síncrona, mas todas as cópias devem seguir o mesmo estado da linha do tempo*" (RAMAKRISHNAN, 2012).

Por fim, o Yahoo! Message Broker auxilia o modelo de consistência de dados. Entre suas atribuições está, ainda de acordo com Cooper et al. (2008), garantir que atualizações não são perdidas até que se verifique que todas as réplicas a aplicaram.

O PNUTS, semelhantemente ao modelo *Knowing* apresentado neste trabalho, utiliza a abordagem *publish/subscribe* para propagação das atualizações. No entanto, como citado acima, ela é feita de modo assíncrono e a consistência é, portanto, relaxada.

Deuteronomy

O sistema de banco de dados Deuteronomy dá suporte à execução de transações no ambiente em nuvem de modo a garantir que não haja conflito de operações concorrentes.

De acordo com Levandoski et al. (2011), o sistema possui dois componentes principais: o TC (*transaction component,* ou componente

transacional), que tem por função garantir o controle de concorrência e recuperação por meio de *undo* e *redo* (desfazer e refazer); e um DC (*data component,* ou componente de dados), responsável pelo acesso físico aos dados. Levandoski et al. (2011) exemplificam a interação desses componentes do seguinte modo:

> Considere uma transação que atualiza duas tabelas: uma tabela de encomendas armazenada em um DC hospedado em um servidor da empresa e uma tabela de pagamentos armazenada em outro DC na nuvem. Um cliente executa esta operação em um único TC, sem especificar a localização física dos dados. O TC realiza todas as operações necessárias para suporte transacional (por exemplo, o registro / bloqueio), e encaminha as operações de atualização de dados para o DC correto (...). Após a conclusão, o TC é responsável por confirmar a transação e garantir as atualizações são estáveis em ambos os DCs

Assim, as transações são executadas por meio da utilização de bloqueios em um nível lógico (isto é, sem que o TC saiba da localização física dos dados).

ElastraS

O ElastraS é um sistema de banco de dados relacional que dá suporte a apenas um subconjunto de operações dos sistemas de banco de dados tradicionais e foi desenvolvido com o intuito de prover garantias transacionais de um modo escalável (DAS et al., 2009).

No tocante à consistência, o ElastraS "*garante a consistência apenas dentro de uma partição do banco de dados e não há nenhuma noção de consistência entre partições*" (DAS et al., 2013).

Isto significa dizer que a consistência de dados é garantida apenas nas operações ocorridas em uma mesma partição, sem garantias de que a leitura realizada em diferentes réplicas retorna o mesmo valor.

Para isso, o ElastraS se utiliza do OTM (*Owning Transaction Manager*), o qual possui acesso exclusivo de leitura e escrita nas partições as quais gerencia, limitando a execução da transação em uma única partição.

EnterpriseDB[24,25]

O EnterpriseDB utiliza duas estratégias para leitura e escrita de dados: *Single-Master*, na qual apenas uma réplica executa as transações e as demais são utilizadas para leitura; e *Multi-Master*, na qual diversas réplicas podem receber transações de escrita e leitura.

No primeiro caso, a replicação dos dados é realizada de modo assíncrono, podendo, portanto, haver inconsistências na leitura. Na estratégia *Multi-Master*, a consistência é eventual.

Síntese dos SGBD em Nuvem

O Quadro 2.2 apresenta uma síntese desses SGBD de modo a organizar as informações anteriormente abordadas. Pode-se verificar que tanto a academia quanto o mercado têm-se preocupado em atender à necessidade do usuário no tocante à consistência de dados, fato este evidenciado pela pluralidade de métodos disponíveis para tal.

Ademais, há sistemas que apostam na interferência do usuário na escolha do nível de consistência utilizado pelo SGBD tanto na academia como em funcionamento efetivo no mercado, como Cassandra, G-Store e o Google Cloud SQL.

Por fim, pode-se verificar que o foco do mercado tem estado em soluções relacionais centralizadas, com suporte à replicação para prover disponibilidade ao usuário, como é o caso do AmazonRDS, Windows Azure e EnterpriseDB. Isso mostra que a realidade do mercado em relação ao modelo de dados relacional ainda não abrange uma das características amplamente divulgadas da nuvem: a escalabilidade.

---

[24] http://www.enterprisedb.com/news-events/press-releases/enterprisedb-s-new-multi-master-replication-ensures-and-expands-data-acce. Acesso: 09/11/2013

[25] http://www.enterprisedb.com/docs/en/9.2/repguide/Postgres_Plus_xDB_Replication_Server_Users_Guide-07.htm. Acesso: 09/11/2013.

Quadro 2.2 - Comparação entre os diferentes SGBD em nuvem.

Nome	Ambiente	Tipo de Garantia de Consistência	Há interferência do usuário?	Síntese do Funcionamento
Amazon SimpleDB	Mercado	Consistência Forte (exceto em caso de sobreposições de leitura ou escrita)	Não	Retorna o resultado das gravações que receberam resposta bem sucedida antes da leitura.
Bigtable	Mercado	Consistência Fraca	Não	A garantia de consistência é dada apenas para dados armazenados na mesma linha e acessados em uma mesma transação
Cassandra	Mercado	Consistência Forte (a depender da combinação das escolhas dos usuários)	Sim	O usuário determina o tipo de consistência entre ALL, ONE e QUORUM
G-Store	Academia	Consistência Forte (para o grupo)	Sim	As aplicações podem selecionar membros para um grupo em tempo de execução permitindo acesso consistente ao conjunto de chaves escolhido
AmazonRDS	Mercado	Consistência Fraca	Não	Permite a utilização de réplicas de leituras, atualizadas assincronamente.
Google Cloud SQL	Mercado	Consistência Forte (para grupos de entidades)	Sim	O usuário deverá organizar em grupos as entidades nas quais deseja garantia de consistência.
Windows Azure	Mercado	Consistência Forte (é um SGBD centralizado)	Não	Não realiza leitura em réplicas distribuídas.
Yahoo! PNUTS	Mercado	Consistência Eventual	Não	Todas as réplicas de um dado registro aplicam as alterações neste registro na mesma ordem
Deuteronomy	Academia	Consistência Forte	Não	Faz uso de bloqueios para garantir a consistência dos dados, o que pode levar a problema de desempenho (RAHIMI e HAUG, 2010).
Elastras	Academia	Consistência Fraca	Não	A consistência de dados é garantida apenas nas operações ocorridas em uma mesma partição
EnterpriseDB	Mercado	Consistência Eventual	Não	Na abordagem *multi-master*, garante a consistência eventual; e na abordagem *single-master*, as réplicas são atualizadas assincronamente.

Fonte: A Autora.

Vale ressaltar ainda, que a lista apresentada não é exaustiva, uma vez que há outros sistemas de banco de dados em nuvem não relacionais (como o Voldemort[26] e o MongoDB[27]) e relacionais (como o StormDB[28], ClearDB[29], entre outros) que não foram aqui detalhados.

A seção seguinte, por sua vez, aborda trabalhos relacionados a este, isto é, métodos, técnicas ou processos para garantia de consistência, os quais serviram de inspiração para o seu desenvolvimento.

2.3. Trabalhos Relacionados

A garantia de consistência de dados é um tema bastante recorrente em sistemas de banco de dados distribuídos tradicionais, fazendo-se presente em diferentes livros sobre o tema, como o de Bell e Grimson (1992), Date (2004), Ray (2009), Rahimi e Haug (2010), Özsu e Valduriez (2011), entre outros.

No que se refere à garantia de consistência em sistemas de banco de dados em nuvem, o tema tem sido abordado discretamente na literatura científica. Apesar disso, é possível elencar diferentes trabalhos que tratam de técnicas e modelos para garantia de consistência de dados (ao contrário da Seção 2.2.5, que aborda Sistemas de Gerenciamento de Bancos de Dados completos, com todas as suas características essenciais implementadas). Esses trabalhos serão discutidos nas seções seguintes.

2.3.1. *An Application-Based Adaptive Replica Consistency for Cloud Storage* - Wang et al. (2010)

O trabalho de Wang et al. (2010), divide a estratégia de consistência em quatro categorias de acordo com a frequência de leitura e escrita. Na categoria C1, a frequência de leitura é alta e a de atualização é baixa; na categoria C2, a frequência de leitura é alta e a de atualização é alta; na categoria C3, a frequência de leitura é baixa e a de atualização é alta; na categoria C4, a frequência de leitura é baixa e a de atualização é baixa. A consistência da réplica

[26] http://www.project-voldemort.com/voldemort/. Acesso: 22/01/2014
[27] http://www.mongodb.org/. Acesso: 22/01/2014.
[28] https://www.stormdb.com/. Acesso: 22/01/2014.
[29] http://www.cleardb.com/. Acesso: 22/01/2014.

é adaptada em tempo de execução, de acordo com a categoria em que esta é classificada.

O modelo considera que a exigência de consistência de uma aplicação é variável em tempo de execução. Por exemplo, em um sistema de leilão: no início do leilão, os dados não são tão importantes para o lance final, por isso a exigência de consistência é baixa. Com o fim do prazo para lances se aproximando, os clientes necessitam cada vez mais dos dados mais recentes para fazer o próximo lance.

Verifica-se, no entanto, que não há nenhuma participação do usuário na escolha do nível de consistência com o qual os dados devem ser tratados. Ademais, como a consistência é tratada em nível de réplica, não sendo possível ao usuário sequer visualizar quais dados têm consistência garantida ou relaxada. Não há informações sobre os sistemas de banco de dados utilizados junto a este modelo.

2.3.2. *Application-Managed Replication Controller for Cloud-Hosted Databases* – Zhao et al. (2012)

Zhao et al. (2012) apresentam um *framework* para replicação de banco de dados no ambiente em nuvem, onde o usuário deve definir um SLA de atualidade dos dados para cada réplica.

O modelo proposto nesse artigo dispõe de três módulos: monitor, controle e ação. O primeiro módulo monitora o atraso de cada réplica (isto é, a diferença de tempo entre os *timestamps* de dados escritos no mestre e na réplica), enviando esses dados ao módulo de controle; este, por sua vez, deverá comparar esse atraso à SLA definida e disparar um comportamento do módulo de ação: Caso uma quantidade de atrasos T seja violada em uma dada réplica, o módulo de ação deverá acionar a réplica geograficamente mais próxima, de modo a equilibrar a carga de trabalho e satisfazer novamente o SLA definido. O módulo de ação poderá também criar novas réplicas.

O modelo de Zhao et al. (2012) trata apenas da atualidade dos dados e não garante a consistência forte, uma vez que o SLA definido pelo usuário

precisa ser violado para que o módulo de ação entre em atividade. Ademais, apesar de dar suporte a banco de dados relacionais, bem como faz o modelo *Knowing*, o processo de definição do SLA pode ser bastante difícil, já que não há nenhum suporte, por parte do sistema, a fim de auxiliar o usuário nesta tarefa.

2.3.3. *Harmony: Towards Automated Self-Adaptive Consistency in Cloud Storage* - Chihoub et al. (2012)

Outro trabalho nesse contexto é o *Harmony*, apresentado no artigo de Chihoub et al. (2012). O *Harmony* tem por objetivo ajustar o nível de consistência adaptativamente de acordo com os requisitos da aplicação definidos pelo usuário (de 0 a 100% de taxa de leitura obsoleta aceitável) e do estado do sistema de armazenamento (como a latência de rede, por exemplo). A partir desses dois parâmetros o *Harmony* aumenta ou diminui o número de réplicas envolvidas em operações de leitura para manter uma baixa fração (possivelmente zero) de leitura obsoleta, melhorando o desempenho das aplicações e mantendo o nível de consistência desejada.

Os testes apresentados mostram que ele reduz a taxa de leitura de dados obsoletos, no entanto, mais uma vez, não há nenhum suporte para que o usuário defina taxa de leitura obsoleta aceitável. Ademais, diferentemente do modelo *Knowing* aqui proposto, o *Harmony* dá suporte a bancos de dados não relacionais, utilizando-se do sistema de banco de dados Cassandra.

2.3.4. *Consistency Rationing in the Cloud: Pay only when it Matters* - Kraska et al. (2009)

Um trabalho bastante citado sobre o tema é o *Consistency Rationing in the Cloud: Pay only when it Matters*, de Kraska et al. (2009), o qual adapta dinamicamente o nível de consistência por meio do monitoramento e da coleta de estatísticas temporais dos dados. Assim como no trabalho de Wang et al. (2010) (Seção 2.3.1), nem todos os dados precisam ser tratados com o mesmo nível de consistência. Deve-se observar que quanto maior é o nível de consistência desejada, maior é o custo associado para obtê-la.

Os dados são divididos em três categorias: (A) refere-se aos dados que necessitam consistência forte – alcançado por meio do bloqueio em duas fases; (B) onde a consistência varia ao longo do tempo; e (C) dados cuja inconsistência pode ser tolerada (utiliza consistência de sessão). Para dados na categoria B, o *Consistency Rationing* calcula o melhor custo possível para executar a transação tendo em vista o *tradeoff* entre consistência e disponibilidade.

O bloqueio distribuído, como o usado para a categoria A, pode levar a um aumento do tempo de resposta, devido à distância entre as réplicas. Ademais, o usuário não tem controle sobre quais dados são tratados ou não com consistência forte. Por fim, este trabalho utiliza o sistema de banco de dados Amazon S3.

2.3.5. *Towards Transactional Data Management over the Cloud* - Tiwari et al. (2010)

O trabalho de Tiwari et al. (2010) apresenta um modelo que visa garantir a consistência dos dados em ambientes geograficamente distribuídos e replicados. Este modelo possui uma arquitetura baseada no gerenciamento de filas, localizadas em cada réplica, e que garantem a execução das transações de acordo com o tempo de chegada das mesmas. O gerenciamento de filas permite a execução das transações na ordem em que ocorrem, garantindo a consistência dos dados. Porém, o modelo proposto por Tiwari et al. (2010) não leva em consideração os diferentes tipos de dados que uma mesma aplicação pode abranger. A utilização dessa estratégia pode aumentar o *delay* na atualização, uma vez que, mesmo que um determinado dado não precise ser tratado de modo consistente, ele deverá ser atualizado imediatamente. Utiliza o sistema de armazenamento Amazon S3.

O Quadro 2.3, mostra um resumo dos trabalhos relacionados.

Quadro 2.3 - Comparativo entre os trabalhos relacionados

Autor(es)	Ano	Tipo de Consistência	Método	Tipo de interferência do usuário	SGBD utilizado
Wang et al.	2010	Consistência forte	Divide a estratégia de consistência em quatro categorias, de acordo com a frequência de leitura e escrita, adaptando o nível de consistência da aplicação em tempo de execução, de acordo com as métricas de leitura e escrita.	Não permite interferência do usuário	Não informado
Zhao et al.	2012	Consistência fraca	Caso uma quantidade de atrasos definida for violada em uma dada réplica com relação ao SLA definido, o módulo de ação aciona a réplica geograficamente mais próxima, de modo a equilibrar a carga de trabalho e satisfazer novamente o SLA definido.	Usuário define um SLA de atualidade dos dados para cada réplica	MySQL
Chihoub et al.	2012	Consistência forte	Ajusta o nível de consistência de acordo com os requisitos da aplicação definidos pelo usuário e do estado do sistema de armazenamento (como a latência de rede, por exemplo).	Usuário define a taxa de leitura obsoleta aceitável (de 0% a 100%)	Cassandra
Kraska et al.	2009	Consistência forte	Adapta dinamicamente o nível de consistência por meio do monitoramento e da coleta de estatísticas temporais dos dados, executando a transação com o menor custo possível.	Não permite interferência do usuário	Amazon S3
Tiwari et al.	2010	Consistência forte	Gerencia filas presentes em cada réplica para garantir a execução das transações na ordem em que elas ocorrem.	Não permite interferência do usuário	Amazon S3

Fonte: A Autora.

Os trabalhos relacionados serviram de inspiração para o modelo proposto no presente trabalho, em especial os de Zhao et al. (2012) e Chihoub et al. (2012), os quais permitem a interferência do usuário na definição do nível de consistência. É possível verificar que esses trabalhos não auxiliam o usuário nesta tarefa. Ademais, o nível de granularidade da consistência dos dados não pode ser verificado.

O modelo *Knowing* aqui proposto minimiza estes problemas, disponibilizando ao usuário um mecanismo simples de definição da consistência, com um nível de granularidade bem definido.

De todos os trabalhos relacionados discutidos, apenas um, o de Zhao et al. (2012), dá suporte a sistemas relacionais, no entanto, ele admite a ocorrência de inconsistências de leitura como parte natural do seu funcionamento. Ao contrário deste trabalho, o modelo *Knowing* visa ajudar garantir a consistência forte, com todos os esforços voltados para dirimir inconsistências de leitura, conforme será mostrado no capítulo seguinte.

2.4. Conclusões

Este capítulo apresentou uma revisão da literatura a fim de fundamentar o presente trabalho. Foram abordados os temas de Banco de Dados em Nuvem (incluindo sua conceituação, características, funcionamento, entre outros aspectos) e Consistência de Dados (abordando o teorema CAP, o PACELC, os tipos de consistência e o modo como diferentes SGBD em nuvem tratam a consistência de dados).

A revisão de literatura mostrou a importância do tema e o modo como ele tem sido abordado sob a ótica de diferentes autores, bem como a necessidade de estudar métodos com vistas a garantir a consistência dos dados em nuvem, ao invés de simplesmente relaxá-la, impondo como razão para isso o teorema CAP.

O próximo capítulo apresenta o modelo *Knowing,* que ajuda a garantir a consistência de dados relacionais em nuvem, descrevendo seus componentes, funcionamento, arquitetura e cenários nos quais a utilização do modelo se mostra adequada.

3. *KNOWING*: UM MODELO PARA AJUDAR A GARANTIR A CONSISTÊNCIA DE DADOS RELACIONAIS EM NUVEM

Este capítulo apresenta a principal contribuição deste trabalho, o modelo *Knowing*, que tem por objetivo ajudar a garantir a consistência dos dados em sistemas de banco de dados relacionais em nuvem. De acordo com o dicionário Webster[30], um modelo pode ser, entre outras coisas:

- Uma cópia geralmente pequena de alguma coisa;
- Um padrão para algo a ser feito;
- Uma descrição ou analogia usada para ajudar a visualizar alguma coisa que não pode ser observada diretamente (como um átomo);
- Um sistema de postulados, dados e inferências apresentadas como uma descrição matemática de uma entidade ou estado de coisas; ou
- Uma projeção teórica de um sistema possível ou imaginário.

Sobre essas definições, Dieste et al. (2002) pontuam que cada uma delas refere-se a um propósito particular dos modelos, isto é, para quê eles são úteis, que resultado sua aplicação vai produzir, entre outros. Os autores destacam ainda que, no desenvolvimento de *software*, os modelos são utilizados para descrever como será o sistema de *software* que resolve um problema, assim como faz o modelo *Knowing*.

O modelo *Knowing* aqui proposto parte do pressuposto de que uma mesma aplicação pode abrigar dados que necessitam de garantia de consistência e outros que não necessitam. Por exemplo, em um sistema de compra de passagens aéreas, a informação referente à quantidade de vagas disponíveis em determinado voo deve ser tratada de modo a garantir que todos os usuários visualizem a mesma informação, a fim de que não haja a venda de passagens excedentes. Nessa mesma aplicação, no entanto, a foto do cliente, parte do seu cadastro, não precisa ser imediatamente atualizada em todas as réplicas.

[30] http://www.merriam-webster.com/dictionary/model. Acesso: 03/03/2014.

A estratégia para lidar com os dois diferentes tipos de dados será dividi-los em grupos:

- Dados que precisam de garantia de consistência devem ser tratados por meio de atualização ansiosa. Este modelo de atualização, no entanto, é diferente dos abordados na literatura (ver Seção 2.3), uma vez que é baseado na comunicação em grupo; e

- Dados que não necessitam de garantia de consistência forte devem ser tratados por meio de técnicas de atualização preguiçosa, como a consistência eventual, as quais foram discutidas anteriormente (Seção 2.2.3).

Para definir o modo de tratamento dos dados, utiliza-se o conhecimento do desenvolvedor sobre sua aplicação: o próprio usuário deverá definir quais tabelas devem ser tratadas com garantia de consistência e quais podem ter a garantia relaxada.

Conforme mostrado no Capítulo 2 deste trabalho, a interferência do usuário na decisão sobre a consistência de dados tem sido a estratégia de diferentes sistemas em nuvem. Apesar disso, percebeu-se uma oportunidade para aperfeiçoar o referido método, uma vez que os sistemas que permitem a interferência do usuário não os auxiliam nesta tarefa. Desse modo, a estratégia de escolha das tabelas que necessitam ou não de garantia de consistência foi determinada pela sua simplicidade e por privilegiar o conhecimento do usuário sobre sua aplicação.

As seções seguintes deste capítulo apresentam a arquitetura e as características do modelo de atualização proposto.

3.1. Modelo de Atualização Ansiosa Baseado em Comunicação em Grupo (*Middleware* Orientado a Mensagem)

Conforme discutido na Seção 2.2.4, uma das estratégias utilizadas pelo modelo de atualização ansiosa tradicional é o *commit* em duas fases (2PC), a

qual pode causar problemas de desempenho. Também nas seções seguintes, (2.2.5 e 2.3), foram abordados diferentes modelos para garantia de consistência, utilizados por diferentes SGBD em nuvem.

Por conseguinte, a partir desses estudos e mediante investigação de seus pontos fortes e suas carências, o modelo aqui proposto foi desenvolvido. O referido modelo dispensa a utilização do *commit* em duas fases, a fim de viabilizar a estratégia de escolha das tabelas.

No lugar do 2PC, o *Knowing* utiliza a comunicação em grupo, por meio de um *Middleware* Orientado a Mensagem (MOM). De acordo com Chappell (2004), o MOM é "*um conceito que envolve a passagem de dados entre aplicações utilizando um canal de comunicação que transporta unidades independentes de comunicação (mensagens)*". Ou seja, o MOM permite a troca de mensagens entre aplicações utilizando estruturas de dados específicas para armazená-las (que podem ser filas ou tópicos).

A abordagem do MOM escolhida para este trabalho foi a ***publish-subscribe***. De acordo com Eugster et al. (2003), no modelo *publish/subscribe:* "*assinantes (subscribers) registram seu interesse em um evento, ou um padrão de eventos, e são subsequentemente notificados, de forma assíncrona, de eventos gerados por aqueles que publicam (publishers)*". Isto é: as aplicações se comunicam por meio de mensagens, que são armazenadas pelos *publishers* em tópicos e retiradas sincronamente pelos interessados nas mesmas, os *subscribers*. Um tópico, por sua vez, é uma estrutura de dados semelhante a uma fila, que se diferencia desta por permitir a existência de mais de um *subscriber*.

No modelo *publish/subscribe* cada tópico possui um assunto e as réplicas subscrevem-se nos tópicos de acordo com o assunto de seu interesse (EUGSTER, 2007). No modelo proposto, cada tópico deverá conter dados de uma tabela específica. Por exemplo, o tópico denominado "topicoCliente" deverá conter todas as mensagens destinadas à tabela "Cliente". A mensagem, no referido modelo, contém uma transação (de inserção, alteração ou remoção).

O armazenamento das mensagens em tópicos permite que o sistema continue em funcionamento mesmo se o *publisher* ou *subscriber* falhar. Isso porque a comunicação ocorre sem que as partes envolvidas no processo se conheçam:

> No modelo *publish/subscribe*, (...) produtores publicam a informação na rede sem conhecer a identidade, localização e o número de *subscribers*. Da mesma forma, os consumidores subscrevem a uma informação específica sem conhecer a identidade, localização e número de *publishers*. (CHEUNG e JACOBSEN, 2010).

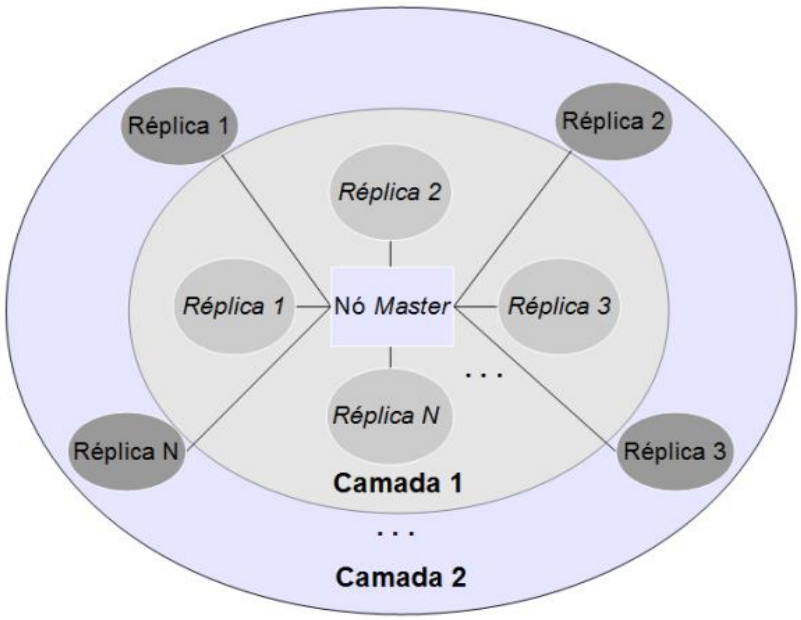
No modelo proposto, para que haja a troca de mensagens, faz-se necessário apenas que o endereço onde o tópico está localizado seja conhecido pelas partes. As demais informações, como a quantidade de remetentes ou destinatários, por exemplo, podem mudar com frequência sem afetar o andamento da comunicação. Essa capacidade de aumentar ou diminuir o número de *subscribers* sem afetar a troca de mensagens viabiliza o particionamento dinâmico de dados. Deste modo, a criação de novas réplicas com os dados particionados poderá ocorrer naturalmente.

Outro benefício do MOM, de acordo com Buschmann et al. (2007), está na capacidade de transmissão de dados do remetente ao destinatário sem que haja bloqueios de espera por resposta. Além disso, as atualizações podem ser realizadas em paralelo – ou seja, cada réplica poderá aplicar a atualização independentemente da outra.

Em uma visão geral, o modelo funciona da seguinte maneira, conforme mostra a Figura 3.1Figura **3.1**: há um único mestre, o nó *master,* que abriga o banco de dados primário (aquele em que as transações irão ocorrer primeiramente). É para este nó que devem ser direcionadas todas as transações; e duas camadas de réplicas, a primeira camada abriga as réplicas com dados de tabelas que necessitam de garantia de consistência. Quando a transação é confirmada no nó *master*, os *subscribers* (isto é, as réplicas da Camada 1) são notificados, devendo aplicá-la em seus bancos de dados na ordem em que as mesmas ocorrem. Todos os *subscribers* deverão aplicar a atualização a fim de que a consistência seja mantida.

Os dados de tabelas que não necessitam de garantia de consistência são direcionados diretamente do nó *master* às réplicas da Camada 2, onde devem ser atualizadas por meio da consistência eventual.

Figura 3.1 - Modelo de replicação para ajudar na garantia de consistência dos dados da Camada 1 e consistência eventual da Camada 2.



Fonte: A Autora.

O sistema *Knowing* é dividido em duas partes. A parte principal, que gerencia as transações realizadas no nó *master,* direcionando-as em formato de mensagem ao tópico adequado, é denominada de ***middleware master***.

Já a parte secundária tem por função receber as mensagens provenientes do *middleware master* e aplicar as transações nelas contidas às réplicas da Camada 1 ou 2. Esta segunda parte do sistema *Knowing* está dividida em ***middleware* da Camada 1** e ***middleware* da Camada 2**, respectivamente**.**

Para que não haja esperas indefinidas, devem ser estabelecidos prazos máximos de espera. Nestes casos, as réplicas não atualizadas deverão ficar indisponíveis para leitura até que sejam recuperadas e as atualizações aplicadas. Deste modo, mesmo que uma ou mais réplicas falhem e não sejam atualizadas, a consistência continua mantida.

Os detalhes da arquitetura e funcionamento do *middleware master* e das Camadas 1 e 2 serão apresentados na seção seguinte.

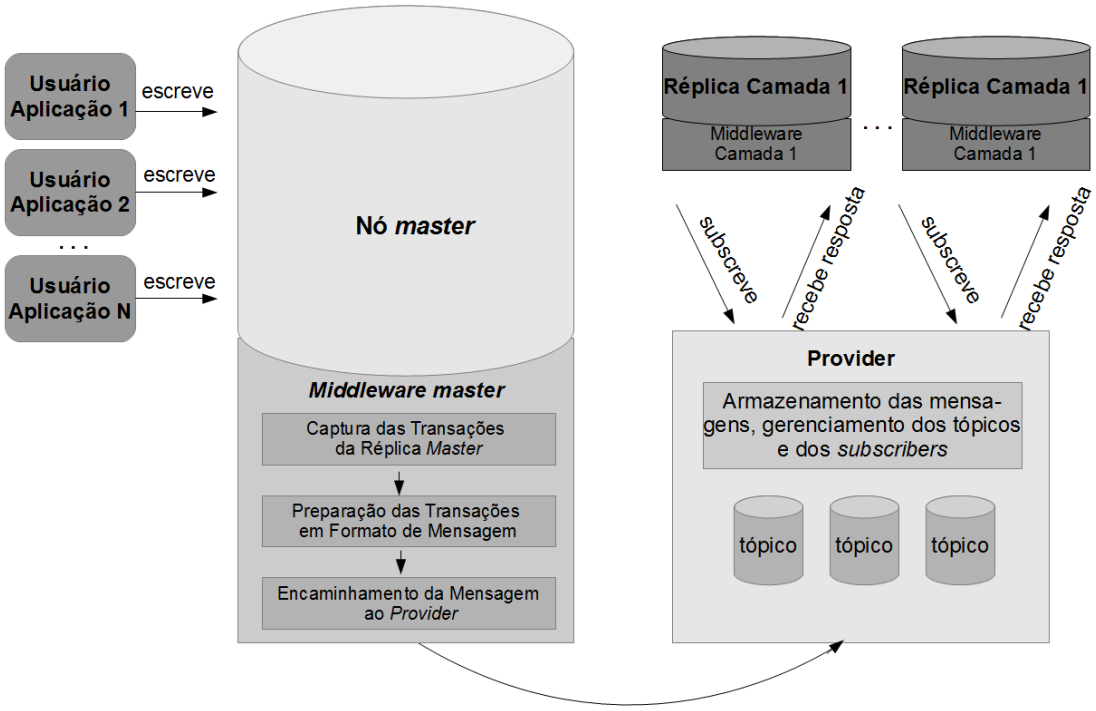
3.2. Arquitetura do *Knowing*: Uma Visão Geral do Modelo

Esta seção tem por objetivo descrever o *Knowing*, explanando em detalhes sua arquitetura e funcionamento. Antes de expor cada uma das partes do *Knowing*, no entanto, convém evidenciar uma característica do nó *master* essencial para o funcionamento do modelo.

O nó *master*, conforme mencionado anteriormente, abriga a réplica primária e deve receber todas as transações e aplicá-las. Após isso, as transações devem ser armazenadas em um *log*, na ordem em que acontecem, a fim de que o *middleware master* possa capturá-las e direcioná-las ao tópico adequado. O modo como este *log* é realizado será descrito juntamente ao protótipo, no Capítulo 4.

A Figura 3.2 ilustra a arquitetura proposta, bem como destaca a interação dos elementos do *middleware master* e o *middleware* da Camada 1*:* Os usuários das aplicações escrevem no nó *master*. Em seguida, o middleware *master* captura essas transações, prepara-as em formato de mensagem e as envia ao *Provider*. Este, por sua vez, armazena as mensagens e gerencia os tópicos e os respectivos *subscribers* de cada tópico. Para isso, ele mantém informações como quais *subscribers* desejam receber a mensagem de um determinado tópico e quais *subscribers* já retiraram cada uma das mensagens, a capacidade de armazenamento dos tópicos e outras informações relevantes. Por fim, as réplicas da Camada 1, que monitoram a chegada das mensagens aos tópicos de seu interesse, deverão então retirá-la e aplicá-la ao seu banco de dados.

Figura 3.2 – Arquitetura do modelo *Knowing*



Fonte: A Autora.

Em seguida são explanados os componentes do modelo *Knowing*.

3.2.1. *Middleware Master*

O *middleware master* tem as seguintes funções: (1) retirar do *log* as transações aplicadas ao nó *master*; (2) organizá-las em formato de mensagem; e (3) encaminhar as mensagens ao *Provider* para armazenamento no tópico adequado.

Para isso, o *middleware master* deverá retirar as mensagens do nó *master* na ordem em que estas ocorrem, a partir do *log* citado anteriormente. De posse da mensagem, esta deverá ser organizada para o formato aceito pelo *Middleware* Orientado a Mensagem.

Observe-se que a confirmação da transação no banco de dados do nó *master* garante a verificação de restrições de consistência e eventuais dependências que possam existir no banco de dados, uma vez que não há

mudanças nas funções-padrão do SGBD utilizado. Desse modo, essas transações podem ser executadas nas outras réplicas, mantendo tais benefícios, ao seguir a ordenação total.

Após retirar a transação, o *middleware master* deverá verificar o destino da mesma. Caso a mensagem seja destinada a uma réplica com dados que não necessitam de garantia de consistência, esta deverá apenas ser direcionada para as réplicas da Camada 2.

Caso o destino seja formado por réplicas com dados em que há necessidade de garantir a consistência, o *middleware master* deverá armazenar a mensagem no tópico adequado. Para isso, o *middleware* deverá utilizar-se do *Provider*, que é o responsável por armazenar as mensagens e gerenciar os tópicos. As réplicas da Camada 1 deverão, para tanto, subscrever-se nos tópicos com dados de suas respectivas tabelas.

Outro ponto a ser observado é o modo como as mensagens chegarão ao destinatário. No *middleware* Orientado a Mensagem, há duas formas de o cliente ser avisado de que há mensagens no tópico: *Pull* e *Push*.

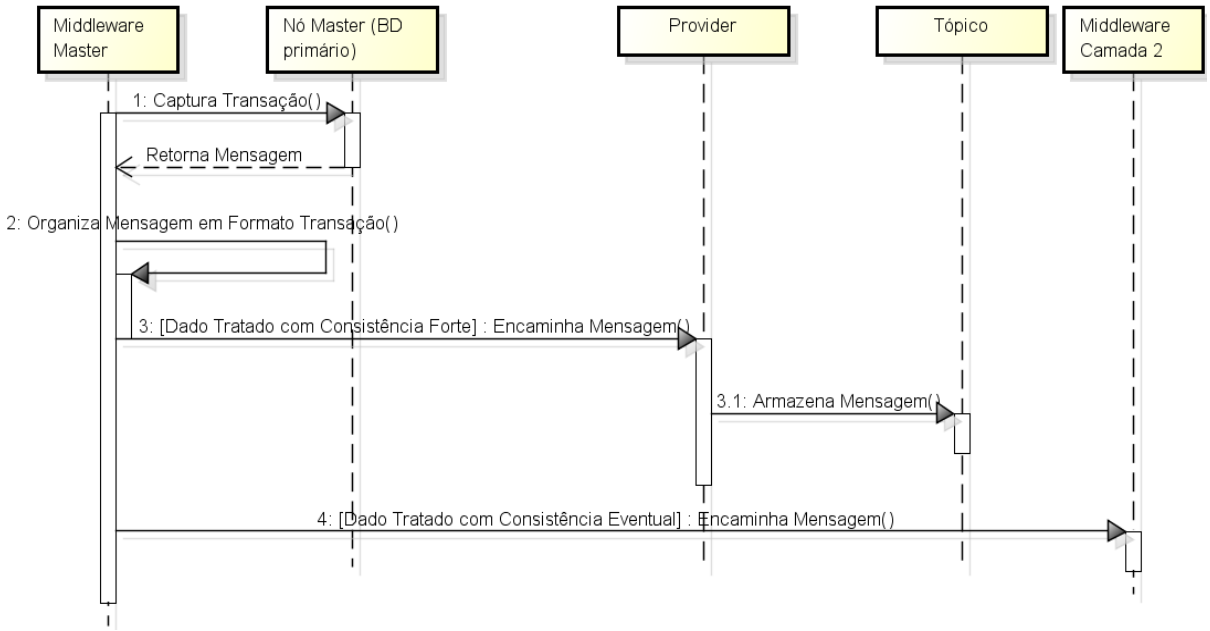
No primeiro, o cliente verifica de tempos em tempos a informação; no segundo, o *Provider* fica responsável por dar a informação ao cliente (CURRY, 2005). O padrão *pull* foi escolhido a fim de distribuir a carga das atualizações. Uma vez que cada réplica deverá retirar sua mensagem, o *Provider* não precisará enviar todas as mensagens ao mesmo tempo, evitando sobrecarrega na rede.

Interação entre os Elementos do *Middleware* Master

A sequência de processos executados para a troca de mensagens é mostrada nos Diagramas de Sequência, desenvolvidos com auxílio da ferramenta *Astah Community*[31]:

[31] http://astah.net/. Acesso: 14/03/2014.

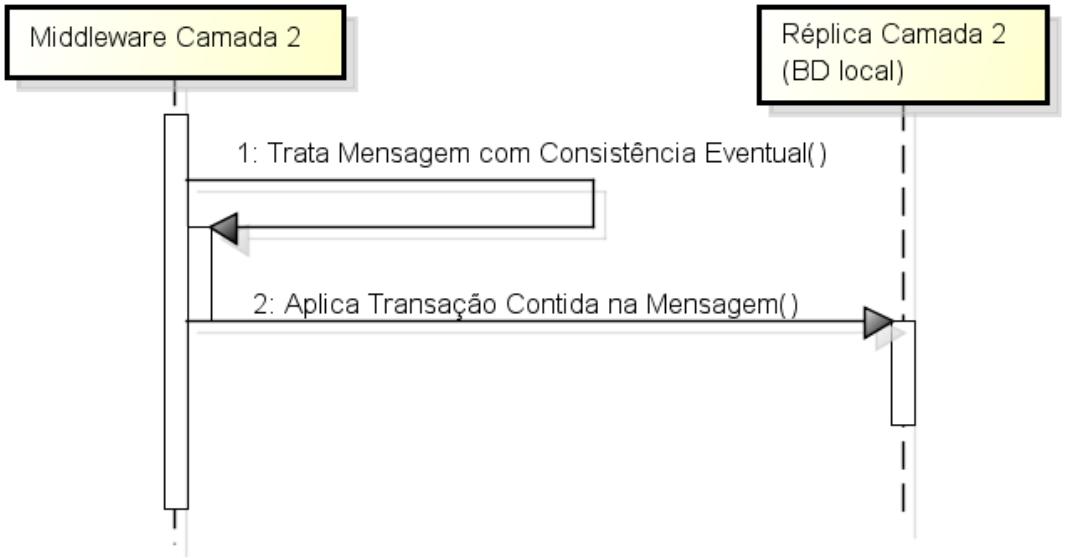
Figura 3.3 - Diagrama de Sequência que mostra o funcionamento do *Middleware Master*



Fonte: A Autora.

O *Middleware Master* captura a transação ocorrida no nó *master* e as organiza em formato de transação (inserção, alteração ou remoção). Dados que necessitam de garantia de consistência são encaminhados ao *Provider,* que armazena a mensagem recebida no devido tópico. Dados que não necessitam de garantia de consistência forte são encaminhados ao *middleware* da Camada 2.

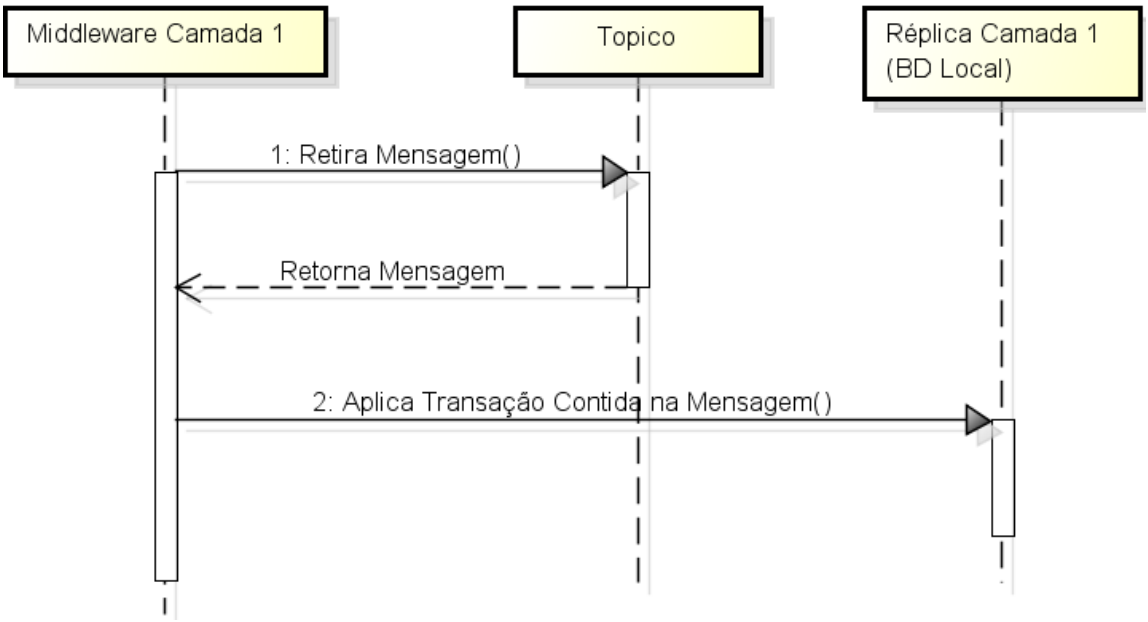
Figura 3.4 - Diagrama de Sequência que mostra o funcionamento do *Middleware* da Camada 2



Fonte: A Autora.

Ao receber a mensagem encaminhada pelo *middleware master*, o *middleware* da Camada 2 deve tratar a mensagem de modo a manter a consistência eventual e, por fim, aplicar a transação contida na mensagem ao seu banco de dados.

Figura 3.5 - Diagrama de Sequência que mostra o funcionamento do *Middleware* da Camada 1



Fonte: A Autora.

Para mensagens que necessitam de garantia de consistência forte, o *middleware* da Camada 1 retira a mensagem do tópico e a aplica ao seu banco de dados.

É sabido que a arquitetura centralizada, com apenas um nó *master,* pode causar problemas de indisponibilidade, por ser esse um ponto único de falha. Para minimizar este problema, o *Provider* poderá ser disposto em um nó diferente do nó *master*. Essa conduta permite que o sistema continue a funcionar para quaisquer leituras mesmo que o nó *master* falhe, uma vez que o *Provider* continua a funcionar, mantendo atualizadas as réplicas da Camada 1.

Ademais, conforme pode ser verificado na Seção 2.2.5, essa arquitetura tem sido utilizada frequentemente na prática pelos SGBD relacionais em nuvem, apesar desse conhecido problema, o que mostra sua aplicabilidade.

Maiores detalhes sobre o funcionamento do *middleware* da Camada 1 e seus aspectos arquiteturais são descritos na seção seguinte.

3.2.2. *Middleware* da Camada 1

O *middleware* da Camada 1 tem as seguintes funções: (1) subscrição nos tópicos de interesse; (2) monitoramento da chegada de novas mensagens no tópico; e (3) Retirada das mensagens e aplicação das transações (contidas nas mensagens) nos seus respectivos bancos de dados.

Para realizar a primeira função, o *middleware* da Camada 1 deverá apenas enviar uma mensagem ao *Provider* contendo o tópico de interesse.

O monitoramento das mensagens, segunda função listada, deverá ser realizado constantemente no sentido do *middleware* da Camada 1 para o *Provider.* Quando uma nova mensagem é armazenada no tópico, o *middleware* da Camada 1 deverá retirar a mensagem do tópico e trazê-la à sua réplica.

Por fim, estas mensagens deverão ser aplicadas no banco de dados das réplicas da Camada 1. Ressalta-se, porém, que esta aplicação deverá seguir o ordenamento FIFO (*First in, First out)*, isto é, a primeira mensagem contida no tópico deverá ser a primeira retirada. Essa ordem deve ser seguida também na aplicação das mensagens nas réplicas-destino.

Toda a comunicação é realizada no sentido da réplica da Camada 1 para o *Provider*. Deste modo, pode-se verificar o total desacoplamento entre *publisher* (que é o *middleware master*) e *subscribers* (que são as réplicas da Camada 1).

A seção seguinte apresenta o *middleware* da Camada 2, sua arquitetura e componentes.

3.2.3. *Middleware* da Camada 2

O *middleware* das réplicas da Camada 2 tem apenas as seguintes funções: (1) receber as transações advindas do *Middleware Master*; e (2) aplicar as transações contidas na mensagem.

Ao receber a mensagem vinda do *middleware master*, esta parte do *Knowing* poderá aplicar quaisquer das técnicas consolidadas de consistência eventual, inclusive as discutidas na Seção 2.2.3 deste trabalho.

A seção seguinte apresenta exemplos da utilização do modelo *Knowing*.

3.3. Como Utilizar o Modelo *Knowing*

Esta seção tem por objetivo apresentar cenários nos quais o modelo *Knowing* pode ser empregado em diferentes sistemas, com vistas a evidenciar a sua utilização. Entre outros exemplos, podem-se citar os seguintes:

Aplicação de vendas de produtos pela Internet - Nesse tipo de aplicação, o usuário poderia selecionar para a garantia de consistência, por exemplo, as tabelas que armazenam a quantidade dos produtos mantidos em estoque e as informações sobre os pagamentos confirmados. Para que não haja vendas de produtos sem saldo em estoque, faz-se necessário garantir a consistência desses dados. Observe-se que quando um pagamento é confirmado, o estoque deve ser imediatamente atualizado, o que exige que ambas as tabelas tenham garantia de consistência. Por outro lado, os dados das sugestões de compras para o usuário, que podem ser do tipo: “quem comprou este produto comprou também este” e os comentários dos usuários sobre um produto comprado, por exemplo, podem ser atualizados por meio da consistência eventual, uma vez que não há prejuízos ao usuário caso essas informações não sejam imediatamente replicadas.

Aplicações Bancárias – Aplicações deste tipo prezam pela garantia de consistência para assegurar a correta realização das transações (saques, pagamentos, consultas). Por conseguinte, dados armazenados na tabela que mantém o saldo do cliente, por exemplo, devem ser tratados de modo consistente. Assim, garante-se que as transações retornam sempre o valor correto. Nesta mesma aplicação, no entanto, os dados de funções secundárias, tal como a caixa de mensagem por meio da qual o banco se comunica com o cliente, pode ser tratada por meio da consistência eventual.

Aplicações de Leilões *Online* – Uma vez que nessas aplicações o lance vencedor pode diferir dos outros em apenas alguns centavos, as tabelas que abrigam os valores dos lances para um determinado produto devem ser atualizadas de modo consistente. Assim, os clientes que desejam comprar o produto sempre visualizam o valor correto do último lance e podem dar sua oferta adequadamente. Os dados de cadastro do cliente, por sua vez, podem ser atualizados por meio da consistência eventual sem prejudicar o andamento do leilão.

Diversas outras aplicações com a mesma estrutura, isto é, que possuem dados que necessitam e outros que não necessitam de garantia de consistência, podem ser atendidas com o uso do modelo *Knowing*, sendo as citadas acima apenas uma ilustração da aplicabilidade do modelo.

Note-se que, uma vez que o usuário detém o conhecimento sobre a aplicação, selecionar quais tabelas necessitam ou não de garantia de consistência torna-se uma tarefa simples. No entanto, apesar da facilidade do método de seleção das tabelas, esta fase do projeto deve ser planejada cuidadosamente, uma vez que a utilização equivocada desta abordagem, por exemplo selecionando incorretamente as tabelas, impacta negativamente na corretude dos dados da aplicação.

3.4. Conclusões

Este capítulo apresentou o *Knowing*, um modelo que ajuda a garantir a consistência de dados armazenados no ambiente em nuvem. Foram explanados os detalhes do funcionamento do modelo destacando, também, como se dá a interferência do usuário na realização da tarefa de garantia da consistência. Adicionalmente, foram expostos todos os aspectos arquiteturais relacionados ao modelo, inclusive descrição do *middleware master* e do *middleware* das Camadas 1 e 2.

O capítulo seguinte deste trabalho apresenta o protótipo desenvolvido a partir do modelo aqui proposto. Outros detalhes do funcionamento do modelo também são explanados a seguir por meio do código-fonte do *middleware*.

4. UM PROTÓTIPO PARA APLICAÇÃO DO MODELO *KNOWING*

Este capítulo tem por objetivo descrever o protótipo que implementa o modelo *Knowing*, desenvolvido na linguagem Java[32], destacando o ambiente, as tecnologias e estratégias utilizadas para o seu desenvolvimento.

4.1. Caracterização do Minimundo

Antes da exposição das etapas de desenvolvimento do sistema *Knowing*, convém apresentar o minimundo criado com o objetivo de testar o protótipo e que também será utilizado com a finalidade didática, para exemplificar as atividades práticas realizadas.

O minimundo desenvolvido é bastante simples, possui apenas 4 (quatro) entidades, e é utilizado apenas para ilustrar um problema aplicável: Um cliente possui id, CPF, nome, telefone e RG. Cada cliente possui apenas um endereço, que contém id, CEP, rua, número, bairro, cidade e estado.

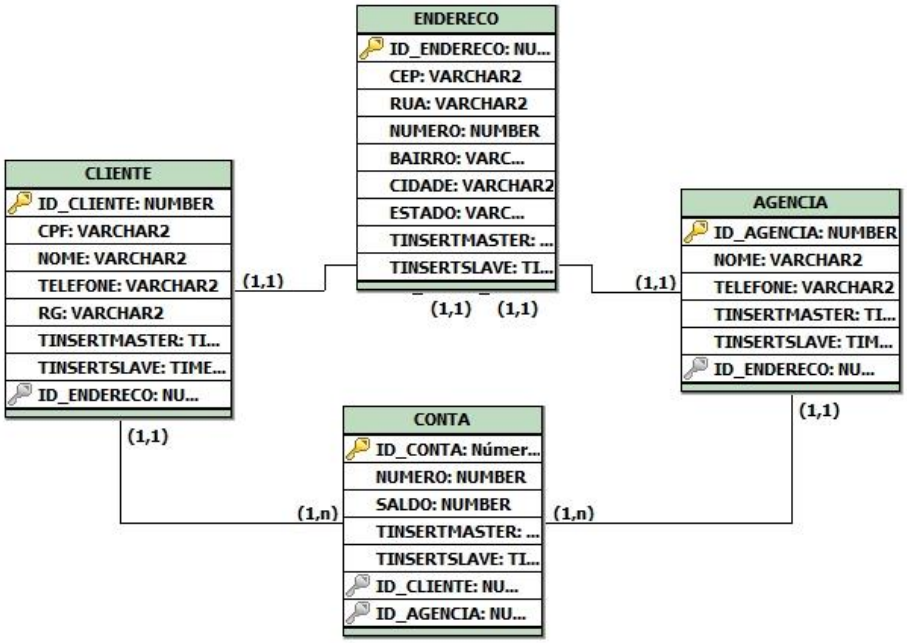
Um cliente pode possuir várias contas, mas uma conta só pode pertencer a uma agência. Conta possui id, número e saldo. Agência possui id, nome, telefone e apenas um endereço.

O tempo em que a tupla foi armazenada no nó *master* e nas réplicas da Camada 1 ou 2 também é armazenado. Para isso, foram criados campos do tipo TIMESTAMP, que deverão armazenar estes dados.

O modelo Relacional deste minimundo pode ser visto na Figura 4.1.

[32] http://www.oracle.com/br/technologies/java/overview/index.html. Acesso: 22/01/2014.

Figura 4.1 - Modelo Relacional do Minimundo descrito. Ferramenta utilizada: BrModelo[33].



Fonte: A Autora.

As seções seguintes têm por objetivo expor o desenvolvimento do protótipo que implementa o modelo *Knowing*, a fim de que o mesmo realize as tarefas descritas no capítulo anterior. A próxima seção, por sua vez, descreve especificamente as configurações do nó *master.*

4.2. Configuração do Nó *Master*

Por nó *master,* entende-se a réplica que irá abrigar o banco de dados primário, ou seja, aquele em que as transações irão ocorrer primeiramente. Nesta réplica ocorrem a recepção e aplicação das transações vindas dos clientes, além da manutenção de um *log* das mesmas.

Para isso, ela deverá possuir um Sistema de Gerenciamento de Banco de Dados instalado e funcionando normalmente. Para esta função, foi o escolhido o SGBD líder de mercado, a saber: Oracle[34].

[33] Disponível em: http://sis4.com/brModelo/download.aspx. Acesso: 29/10/2013
[34] http://www.oracle.com. Acesso: 22/01/2014

Uma vez instalado o SGBD e funcionando normalmente, faz-se necessário realizar o *log* das transações confirmadas no nó *master*. Esta tarefa é descrita na seção seguinte.

4.2.1. Criação do *Log* do Nó *Master*

Os procedimentos realizados para a criação do *log* no nó *master* são parte essencial do modelo *Knowing*, uma vez que cada transação confirmada no nó *master* deverá ser armazenada em uma fila simples para posterior utilização do *middleware master*.

Com esta finalidade, a tecnologia *Oracle Advanced Queueing* (AQ - em português, Enfileiramento Avançado) foi utilizada em conjunto a *triggers* e *procedures.* O Oracle AQ trata-se de um conjunto de funcionalidades dos pacotes DBMS_AQADM e DBMS_AQ, integrados ao sistema de banco de dados, que permite a utilização de filas de mensagens, de modo que estas podem ser salvas de maneira persistente e retiradas da fila na ordem em que foram armazenadas[35].

Ressalta-se, no entanto, que a existência do modelo *Knowing* não está condicionada ao SGBD Oracle, uma vez que a criação do *log* pode ser realizada em outros Sistemas de Banco de Dados. O sistema MySQL, por exemplo, implementa esta funcionalidade por meio do Q4M[36] (*Queue for* MySQL – em português, Fila para MySQL).

Outrossim, esta funcionalidade pode ser implementada por meio de linguagens de programação. Entretanto, uma vez que o Oracle possui suporte nativo ao gerenciamento de filas, optou-se por fazer uso do mesmo. Desse modo, as etapas descritas a partir de então utilizam a sintaxe do sistema Oracle, mas podem ser realizadas analogamente em outro SGBD de interesse.

Em linhas gerais, o *log* funciona da seguinte forma: as tabelas que necessitam de garantia de consistência possuem uma *trigger* associada, que é disparada a cada INSERT, UPDATE ou DELETE. A *trigger* chama um

[35] http://docs.oracle.com/cd/E11882_01/server.112/e11013/aq_intro.htm#ADQUE2419. Acesso: 28/10/2013.
[36] http://q4m.github.io/. Acesso: 27/01/2014.

procedimento que enfileira a mensagem em uma tabela, armazenando o conteúdo de todos os campos da tabela que sofreu a alteração.

As mensagens enfileiradas só podem ser retiradas da fila seguindo a ordem FIFO (*first in, first out)*. As seguintes atividades são necessárias à criação do sistema de enfileiramento:

a) Especificação de um tipo de dado complexo, que irá armazenar o conteúdo da mensagem. Neste caso, foi definido o tipo EVENT_MSG_TYPE, criado de modo a abrigar dados de todas as tabelas do minimundo. Para isso, os seguintes campos foram definidos: ID_MSG, DATA_HORA (data e hora em que a transação foi efetivada no nó *master*), ACAO (inserção, atualização ou remoção), TABELA (nome da tabela alterada), CAMPO_N (nome do campo da tabela), VALOR_N (novo valor do campo) e TIPO_N (tipo do dado do campo). Onde N é o número que indica a ordem do campo na tabela.

O valor de N é igual à quantidade de campos da entidade com o maior número de atributos e tem o objetivo de abrigar dados de todas as tabelas do minimundo.

A Figura 4.2 mostra o código-fonte do tipo complexo EVENT_MSG_TYPE.

Figura 4.2 – Código para criação do tipo de dado complexo EVENT_MSG_TYPE

```
CREATE OR REPLACE TYPE TESTE.EVENT_MSG_TYPE AS OBJECT (
   ID_MSG   NUMBER, ACAO VARCHAR2(50), TABELA VARCHAR(50),
   CAMPO_1 VARCHAR2(50), VALOR_1 VARCHAR2(50), TIPO_1 VARCHAR2(50),
   CAMPO_2 VARCHAR2(50), VALOR_2 VARCHAR2(50), TIPO_2 VARCHAR2(50),
   CAMPO_3 VARCHAR2(50), VALOR_3 VARCHAR2(50), TIPO_3 VARCHAR2(50),
   CAMPO_4 VARCHAR2(50), VALOR_4 VARCHAR2(50), TIPO_4 VARCHAR2(50),
   CAMPO_5 VARCHAR2(50), VALOR_5 VARCHAR2(50), TIPO_5 VARCHAR2(50),
   CAMPO_6 VARCHAR2(50), VALOR_6 VARCHAR2(50), TIPO_6 VARCHAR2(50),
   CAMPO_7 VARCHAR2(50), VALOR_7 VARCHAR2(50), TIPO_7 VARCHAR2(50),
   CAMPO_8 VARCHAR2(50), VALOR_8 VARCHAR2(50), TIPO_8 VARCHAR2(50),
   CAMPO_9 VARCHAR2(50), VALOR_9 VARCHAR2(50), TIPO_9 VARCHAR2(50)
);
```

Fonte: A Autora.

b) Criação da tabela de enfileiramento, a partir do tipo complexo, para armazenar as transações ocorridas**.** Há apenas uma tabela para armazenar

todas as transações. A Figura 4.3 mostra o código utilizado para a criação da tabela EVENT_QUEUE_TAB.

Figura 4.3 - Código para criação da tabela de enfileiramento EVENT_QUEUE_TAB

```
BEGIN
 DBMS_AQADM.CREATE_QUEUE_TABLE (
   QUEUE_TABLE        =>  'TESTE.EVENT_QUEUE_TAB',
   QUEUE_PAYLOAD_TYPE => 'TESTE.EVENT_MSG_TYPE');
END;
```

Fonte: A Autora.

c) Criação e inicialização da fila. Por fim, é necessário criar a fila que irá gerenciar a tabela de enfileiramento. A Figura 4.4 mostra os códigos de criação e inicialização da fila.

Figura 4.4 – Código para criação e posterior inicialização da fila EVENT_QUEUE

```
BEGIN
DBMS_AQADM.CREATE_QUEUE (
   QUEUE_NAME             =>  'TESTE.EVENT_QUEUE',
   QUEUE_TABLE            => 'TESTE.EVENT_QUEUE_TAB');
END;

BEGIN
DBMS_AQADM.START_QUEUE (
   QUEUE_NAME             =>  'TESTE.EVENT_QUEUE',
   ENQUEUE                => TRUE
);
END;
```

Fonte: A Autora.

Após a criação do sistema de filas, já é possível realizar o enfileiramento das transações, que é desencadeado por meio de *triggers.* A Figura 4.5, mostra o código-fonte da *trigger* associada à tabela TB_CONTA.

A *trigger* é executada a cada transação ocorrida na referida tabela e chama a *procedure* P_ENFILEIRAMENTO, passando como parâmetro os valores de cada um dos campos da tabela.

Observe-se que os dados são armazenados de acordo com o tipo de operação, sendo os mesmos posteriormente tratados no código da aplicação para o formato da operação de origem.

Isto é, os dados são armazenados campo a campo e, tratados no código da aplicação, voltam ao formato de inserção, alteração ou remoção. Para a operação de alteração, os campos que sofreram mudanças são marcados a fim de que apenas estes sejam incluídos na transação final.

Figura 4.5 - *Trigger* da tabela de contas, que enfileira as transações ocorridas nesta tabela.

```
CREATE OR REPLACE TRIGGER TESTE.TRIG_ENFILEIRA_CONTA AFTER INSERT OR UPDATE OR DELETE ON TESTE.TB_CONTA FOR EACH ROW
DECLARE
   id_conta varchar2(50);
   num_conta varchar2(50);
   saldo varchar2(50);
   id_cliente varchar2(50);
   id_agencia varchar2(50);
   tinsertmaster timestamp(6);
   tinsertslave varchar2(50);

BEGIN
  IF(INSERTING) THEN
      TESTE.P_ENFILEIRAMENTO (SEQ_LOG_ID.NEXTVAL, 'INSERT', 'TB_CONTA', 'ID_CONTA', :NEW.ID_CONTA, 'NUMBER',
                              'NUM_CONTA', :NEW.NUM_CONTA, 'NUMBER', 'SALDO', :NEW.SALDO, 'NUMBER', 'ID_CLIENTE', :NEW.ID_CLIENTE,
                              'NUMBER', 'ID_AGENCIA', :NEW.ID_AGENCIA, 'NUMBER', 'TINSERTMASTER',
                              LOCALTIMESTAMP, 'TIMESTAMP', 'TINSERTSLAVE', :NEW.TINSERTSLAVE, 'LOCALTIMESTAMP' ,'', '', '', '', '');
  ELSIF (UPDATING) THEN

    IF (:NEW.ID_CONTA = :OLD.ID_CONTA) THEN
       id_conta := '*' || TO_CHAR(:NEW.ID_CONTA);
    ELSE
       id_conta := :NEW.ID_CONTA;
    END IF;

  IF (:NEW.NUM_CONTA = :OLD.NUM_CONTA) THEN
      num_conta := '*' || TO_CHAR(:NEW.NUM_CONTA);
  ELSE
      num_conta := :NEW.NUM_CONTA;
  END IF;

  IF (:NEW.SALDO = :OLD.SALDO) THEN
      saldo := '*' || TO_CHAR( :NEW.SALDO);
  ELSE
       saldo := :NEW.SALDO;
  END IF;

  IF (:NEW.ID_CLIENTE = :OLD.ID_CLIENTE) THEN
      id_cliente := '*' || TO_CHAR (:NEW.ID_CLIENTE);
  ELSE
      id_cliente := :NEW.ID_CLIENTE;
  END IF;
  IF (:NEW.ID_AGENCIA = :OLD.ID_AGENCIA) THEN
      id_agencia := '*' || TO_CHAR(:NEW.ID_AGENCIA);
  ELSE
     id_agencia := :NEW.ID_AGENCIA;
  END IF;

    tinsertslave := 'TO_TIMESTAMP(LOCALTIMESTAMP)';

        TESTE.P_ENFILEIRAMENTO (SEQ_LOG_ID.NEXTVAL, 'UPDATE', 'TB_CONTA', 'ID_CONTA', ID_CONTA, 'NUMBER',
                              'NUM_CONTA', NUM_CONTA, 'NUMBER', 'SALDO', SALDO, 'NUMBER', 'ID_CLIENTE', ID_CLIENTE, 'NUMBER',
                              'ID_AGENCIA', ID_AGENCIA, 'NUMBER', 'TINSERTMASTER', LOCALTIMESTAMP, 'TIMESTAMP',
                              'TINSERTSLAVE', TINSERTSLAVE, 'LOCALTIMESTAMP', '', '', '', '', '');

 ELSIF (DELETING) THEN
        TESTE.P_ENFILEIRAMENTO (SEQ_LOG_ID.NEXTVAL, 'DELETE', 'TB_CONTA', 'ID_CONTA', :OLD.ID_CONTA, 'NUMBER',
                               'NUM_CONTA', :NEW.NUM_CONTA, 'NUMBER', 'SALDO', :NEW.SALDO, 'NUMBER',
                               'ID_CLIENTE', :NEW.ID_CLIENTE, 'NUMBER', 'ID_AGENCIA', :NEW.ID_AGENCIA, 'NUMBER',
                               'TINSERTMASTER', LOCALTIMESTAMP, 'TIMESTAMP', 'TINSERTSLAVE', :NEW.TINSERTSLAVE,
                               'LOCALTIMESTAMP', '', '', '', '', '');

END IF;
END;
```

Fonte: A Autora.

A *procedure* P_ENFILEIRAMENTO, como seu nome sugere, é responsável por enfileirar os dados passados por parâmetro pelas *triggers*, fazendo uso do pacote DBMS_AQ. Deste modo, o objeto complexo EVENT_MSG_TYPE é armazenado com os valores dos parâmetros.

Observe-se que há apenas uma *procedure* para enfileiramento, que atende a todas as tabelas. A Figura 4.6 mostra o código-fonte da *procedure* P_ENFILEIRAMENTO.

Figura 4.6 - Código da *procedure* P_ENFILEIRAMENTO.

```
CREATE OR REPLACE PROCEDURE TESTE.P_ENFILEIRAMENTO
(
   ID_MSG   NUMBER, ACAO VARCHAR2, TABELA VARCHAR2,
   CAMPO_1 VARCHAR2, VALOR_1 VARCHAR2, TIPO_1 VARCHAR2,
   CAMPO_2 VARCHAR2, VALOR_2 VARCHAR2, TIPO_2 VARCHAR2,
   CAMPO_3 VARCHAR2, VALOR_3 VARCHAR2, TIPO_3 VARCHAR2,
   CAMPO_4 VARCHAR2, VALOR_4 VARCHAR2, TIPO_4 VARCHAR2,
   CAMPO_5 VARCHAR2, VALOR_5 VARCHAR2, TIPO_5 VARCHAR2,
   CAMPO_6 VARCHAR2, VALOR_6 VARCHAR2, TIPO_6 VARCHAR2,
   CAMPO_7 VARCHAR2, VALOR_7 VARCHAR2, TIPO_7 VARCHAR2,
   CAMPO_8 VARCHAR2, VALOR_8 VARCHAR2,TIPO_8 VARCHAR2,
   CAMPO_9 VARCHAR2, VALOR_9 VARCHAR2, TIPO_9 VARCHAR2
)
IS
  l_enqueue_options     DBMS_AQ.enqueue_options_t;
  l_message_properties  DBMS_AQ.message_properties_t;
  l_message_handle      RAW(16);
  l_event_msg           TESTE.event_msg_type;
BEGIN
  l_event_msg := TESTE.event_msg_type
  (
   ID_MSG, ACAO, TABELA, CAMPO_1, VALOR_1, TIPO_1, CAMPO_2, VALOR_2, TIPO_2,
   CAMPO_3, VALOR_3, TIPO_3, CAMPO_4, VALOR_4, TIPO_4, CAMPO_5, VALOR_5, TIPO_5, CAMPO_6, VALOR_6,
   TIPO_6, CAMPO_7, VALOR_7, TIPO_7, CAMPO_8, VALOR_8, TIPO_8, CAMPO_9, VALOR_9, TIPO_9
  );

  DBMS_AQ.enqueue(queue_name         => 'TESTE.event_queue',
                  enqueue_options    => l_enqueue_options,
                  message_properties => l_message_properties,
                  payload            => l_event_msg,
                  msgid              => l_message_handle);
END;
```

Fonte: A Autora.

Após execução destas etapas, todas as transações realizadas nas tabelas que necessitam de garantia de consistência são armazenadas em *log*. Deste modo, finaliza-se a configuração do nó *master*.

A próxima seção deste trabalho explana a configuração das réplicas da Camada 1.

4.3. Configuração das Réplicas da Camada 1

A réplica da Camada 1 é responsável apenas por receber as transações enfileiradas e aplicá-las ao seu banco de dados. Por essa razão, a configuração dessas réplicas é mais simples que a do nó *master.*

Desse modo, faz-se necessário apenas um Sistema de Gerenciamento de Banco de Dados Oracle em funcionamento e o serviço de mensagens para que as réplicas desta camada executem sua função.

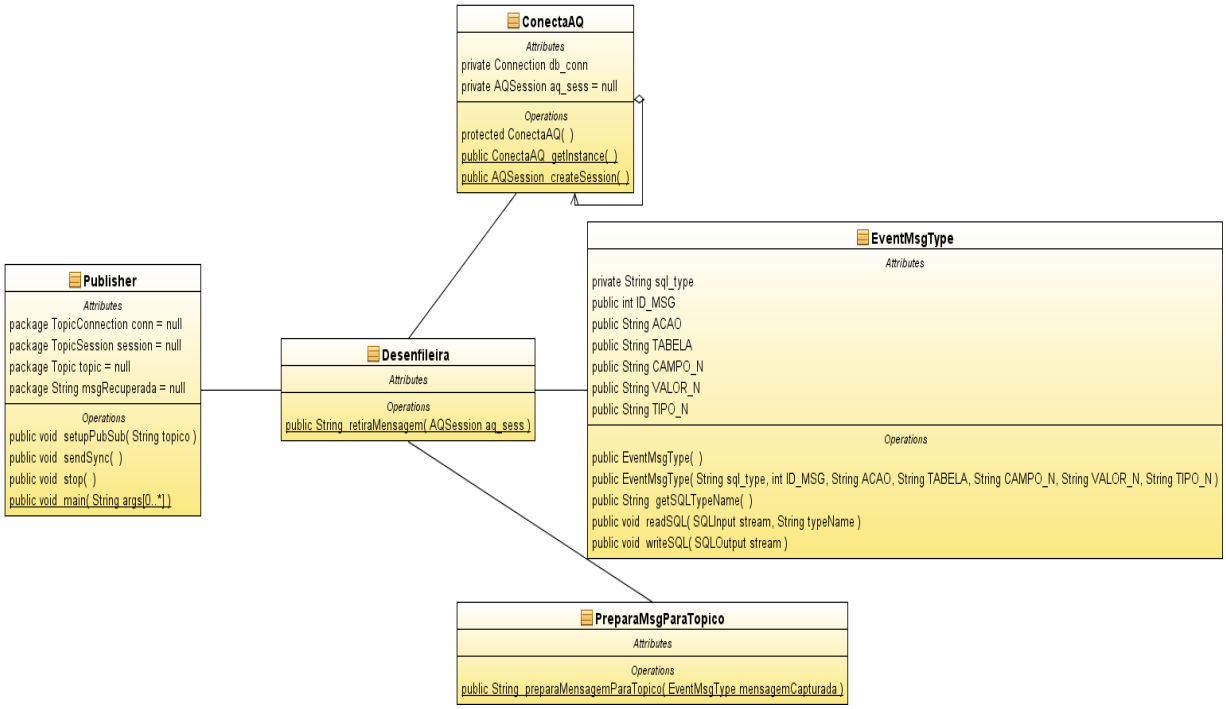
A seção seguinte explana os detalhes do funcionamento do *middleware master*, a ser executado no nó *master*.

4.4. *Middleware Master*

O *middleware master* tem por função capturar as transações ocorridas no nó *master*, as quais estão armazenadas em *log*, e encaminhá-las ao tópico adequado. Os detalhes de desenvolvimento do referido *middleware* será discutido nessa seção.

O *middleware master* pode ser visto no Diagrama de Classes da Figura 4.7, que mostra a estrutura e a relação entre suas classes:

Figura 4.7 - Diagrama de Classes do *Middleware Master*



Fonte: A Autora.

A classe Desenfileira se conecta ao nó *master* por meio da Classe ConectaAQ, desenfileira a mensagem no formato EVENT_MSG_TYPE, e a retorna em formato de transação, fazendo uso da classe PreparaMsgParaTopico. Isto é, um objeto do tipo EVENT_MSG_TYPE é retirado da fila e organizado no formato de inserção, alteração ou remoção. Desse modo, ao receber a mensagem, a réplica de destino pode executar imediatamente a transação.

A Figura 4.8 mostra o código comentado da referida classe:

Figura 4.8 - Classe Desenfileira, que retira as mensagens do nó *master* e retorna uma mensagem em formato de transação.

```
public class Desenfileira {

  public static String retiraMensagem(AQSession aq_sess)
                throws AQException, SQLException, ClassNotFoundException
           {
             Connection           db_conn   = null;
             AQQueue              queue     = null;
             AQMessage            message   = null;
             AQObjectPayload      payload   = null;
             AQDequeueOption      dq_option = null;
             EventMsgType         emt2 = null;
             Hashtable            map;
             String               msgTopico = null;

             //Aqui chama o método de ConectaAQ que retorna uma sessao AQ - aq_sess
             db_conn = ((AQOracleSession)aq_sess).getDBConnection();

             /* Retorna o objeto Queue, onde o primeiro argumento é o usuário e o segundo o nome da fila*/
             queue = aq_sess.getQueue("TESTE", "EVENT_QUEUE");

             //Registra a classe EventMsgType (que corresponde ao tipo EventMsgType)
             try
             {
               map  = (java.util.Hashtable)(((OracleConnection)db_conn).getTypeMap());
               map.put("TESTE.EVENT_MSG_TYPE", Class.forName("beans.EventMsgType"));
             }
             catch(Exception ex)
             {
               System.out.println("Tipo incorreto: " + ex);
             }


             /* Desenfileira uma mensagem da event_queue */

             dq_option = new AQDequeueOption();

             message = queue.dequeue(dq_option, Class.forName("beans.EventMsgType"));
             payload = message.getObjectPayload();
             emt2 = (EventMsgType) payload.getPayloadData();
             db_conn.commit();

             msgTopico  = preparaMensagemParaTopico(emt2);

             return msgTopico;
           } }
```

Fonte: A Autora.

Por fim, o *middleware master* se conecta ao *Provider* e envia a mensagem ao tópico. A Figura 4.9 mostra o método sendSync(), da classe Publisher, que realiza esta tarefa.

Figura 4.9 - Método sendSync(), da classe Publisher, que envia a mensagem desenfileirada para o tópico correto.

```
public void sendSync()
    throws JMSException, NamingException
{

   for (;;) {
    String topico = null;

    //Desenfileira a mensagem, conectando-se à fila do Oracle
    AQSession  aq_sess = null;
    try
          {
             aq_sess = ConectaAQ.createSession();
             msgRecuperada = Desenfileira.retiraMensagem(aq_sess);
          }catch (Exception ex)
          {
             System.out.println("Erro: " + ex);
             ex.printStackTrace();
          }


   if (msgRecuperada.contains("TB_CONTA"))
        topico = "jms/topicoConta";
   else if (msgRecuperada.contains("TB_CLIENTE"))
       topico="jms/topicoCliente";
   else if (msgRecuperada.contains("TB_AGENCIA"))
       topico="jms/topicoAgencia";
   else if (msgRecuperada.contains("TB_ENDERECO"))
       topico="jms/topicoEndereco";

     // Realiza as conexões com o tópico de destino
    setupPubSub(topico);

    // Envia a mensagem recuperada ao tópico
    TopicPublisher send = session.createPublisher(topic);
    TextMessage tm = session.createTextMessage(msgRecuperada);
    send.publish(tm);
    System.out.println("Mensagem Recuperada" +  tm.getText() +  topico);
    send.close();
   } //fim do for
}
```

Fonte: A Autora.

Até então, o desenvolvimento do protótipo envolveu apenas a linguagem de programação Java e os pacotes de enfileiramento do SGBD Oracle.

Para a inserção (Figura 4.9) e retirada das mensagens nos tópicos, além do gerenciamento dos mesmos, utilizou-se a tecnologia *Java Message Service*[37] (JMS, em português, Serviço de Mensagens Java).

O JMS é uma API para criar, enviar, receber e ler mensagens em *Middlewares* Orientados a Mensagem (SUN, 2002). A seção seguinte, por sua vez, apresenta em detalhes o *middleware* da Camada 1.

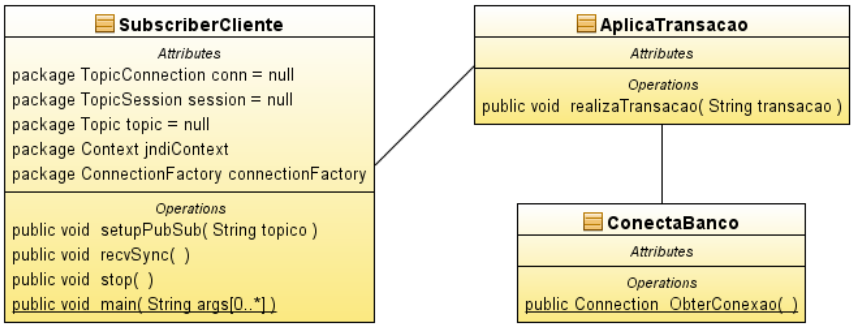
4.5. *Middleware* Camada 1

O *middleware* da Camada 1 deverá retirar a mensagem do tópico e aplicá-la ao seu banco de dados, fazendo uso da atualização ansiosa para ajudar a garantir a consistência dos dados. Para isso, cada um dos *subscribers* se conecta ao tópico de interesse por meio de um código identificador único.

Desse modo, mesmo que a réplica falhe, após recuperar-se, ela receberá todas as mensagens que o tópico recebeu durante o tempo em que esteve sem comunicação. Cada *subscriber* recebe mensagens de um tópico. Para que uma réplica receba mensagens de mais de um tópico, pode-se criar um novo *subscriber* com as configurações do respectivo tópico.

O Diagrama de Classes do *Middleware* da Camada 1 é mostrado na Figura 4.10:

Figura 4.10 - Diagrama de Classes do Middleware da Camada 1



Fonte: A Autora.

[37] http://java.sun.com/products/jms/. Acesso: 29/10/2013

Conforme o Diagrama de Classes da Figura 4.10, a classe SubscriberCliente se conecta ao *Provider* e retira a mensagem do tópico Cliente. Em seguida, aplica a transação no banco de dados da réplica da Camada 1.

A Figura 4.11 mostra o código-fonte da classe SubscriberCliente, que realiza todas essas tarefas na tabela de Clientes. Na imagem, mostra-se o método recvSync(),que realiza as configurações necessárias para conectar-se ao tópico e posteriormente recebe e aplica as mensagens.

Figura 4.11 – Método recvSync, da classe SubscriberCliente.

```
public void recvSync() throws JMSException

{
    AplicaTransacao aplicar = new AplicaTransacao();
    String topico = "jms/topicoConta";
    System.out.println("Início da Recepção de mensagens");

    // realiza as configurações
    setupPubSub(topico);

    TopicSubscriber recv;
    Message msg = null;

        recv = session.createDurableSubscriber(topic, "subsc");

        for (;;) {

            msg = recv.receive();

    if (msg == null) {
        System.out.println("Tempo de espera máximo atingido");
    } else {
        String replyString = null;
        try {
            replyString = ((TextMessage) msg).getText();
        } catch (JMSException ex) {
            ex.printStackTrace();
        }
        aplicar.realizaTransacao(replyString);
        System.out.println("msg: " + replyString );
    }

    }   //Fim do for
}
```

Fonte: A Autora.

Um ponto importante a ser destacado no código é a utilização do método receive(), que retira sincronamente a mensagem armazenada no tópico (SUN, 2002).

4.6. *Middleware* Camada 2

Esta parte do *middleware* deverá estar apta a receber e aplicar as transações advindas do *middleware master*.

Uma vez que esta camada lida com dados que não necessitam de garantia de consistência, quaisquer das técnicas de consistência eventual pode ser utilizada para implementar o *middleware* da Camada 2. Estando o desenvolvimento e utilização de tais técnicas consolidadas, não fará parte do protótipo seu desenvolvimento.

4.7. Utilização do Protótipo em outros SGBD

Conforme mencionado anteriormente, o protótipo aqui desenvolvido pode ser utilizado em outros Sistemas de Gerenciamento de Banco de Dados além do Oracle. Para isso, algumas alterações devem ser realizadas, a saber:

- Alterar o *log* do nó *master* – O *log* do nó *master* foi desenvolvido utilizando os pacotes nativos da Oracle e, para que outros SGBD sejam utilizados, faz-se necessário alterá-lo. Conforme dito anteriormente, o *log* pode ser desenvolvido no MySQL com o utilitário Q4M. Caso o SGBD escolhido não possua um serviço de filas este pode ser programado obedecendo aos seguintes requisitos: (1) Permitir especificação de tipos complexos; e (2) Enfileirar e retirar as mensagens na ordem FIFO. A procedure P_ENFILEIRAMENTO deverá ser adequada para chamar corretamente o serviço de filas;
- Ajustar a classe Desenfileira no *middleware master* - A mudança do serviço de filas implica em mudar a forma de retirar as mensagens da mesma, o qual deve se adequar à sintaxe do novo serviço.

As demais partes do protótipo não necessitam de alterações, uma vez que não são dependentes do Sistema de Banco de Dados utilizado.

4.8. Conclusões

Este capítulo apresentou o protótipo que implementa o modelo para ajudar a garantir a consistência de dados em nuvem, *Knowing*. Foram explanadas todas as etapas necessárias para o desenvolvimento do protótipo, destacando todos os aspectos técnicos em detalhes, desde o processo de enfileiramento até o desenvolvimento do *middleware master* e *middleware* da Camada 1.

O capítulo seguinte deste trabalho apresenta os experimentos realizados com o objetivo de avaliar o protótipo desenvolvido, bem como o resultado da experimentação.

5. EXPERIMENTOS

Este capítulo apresenta os testes executados por meio do protótipo do sistema *Knowing*, os quais foram realizados em um ambiente controlado com o objetivo de verificar sua efetividade na garantia de consistência dos dados.

5.1. Ambiente de Testes

O ambiente de testes foi preparado fazendo uso de 3 (três) máquinas virtuais, sendo uma delas o nó *master* e as duas restantes *subscribers* (Camada 1) as quais foram configuradas como segue[38].

5.1.1. Nó *Master*

Como dito anteriormente, o Oracle foi o SGBD escolhido para o desenvolvimento do protótipo do modelo *Knowing.*

A versão do sistema de banco de dados instalada foi a mais recentemente lançada: Oracle Enterprise 12C (*release* 12.1.0.1.0), que está disponível para *download* no site da empresa[39]. Esta versão foi projetada especialmente para gerenciar bancos de dados na nuvem e possui características específicas desse modelo, como a capacidade de possuir multi-inquilinos, alta disponibilidade e escalabilidade, entre outras[40].

Para sua instalação, foi utilizada uma máquina virtual com 70GB de HD, 2GB de memória RAM e 4GB de *swap*. O *software* utilizado para a tarefa de virtualização foi o *VMWare Player*[41] (5.0.1).

O SGBD foi instalado sobre o sistema operacional Oracle Linux (*release* 6.3.0.0.0 para sistemas 64bits). Este sistema foi desenvolvido pela empresa

[38] Observe-se que todo *software* citado para o desenvolvimento do sistema *Knowing* foi usado na mesma versão em todas as partes do protótipo.
[39] http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html. Acesso: 28/10/2013.
[40] http://www.oracle.com/br/corporate/features/database-12c/index.html. Acesso: 28/10/2013
[41] http://www.vmware.com/go/downloadplayer/. Acesso: 28/10/2013.

especialmente para dar suporte às necessidades dos sistemas de banco de dados da marca. Ele é baseado no Red-Hat e distribuído livremente[42].

Sistema de Mensagens (*Provider*)

O *Provider* tem por função armazenar as mensagens e gerenciar os tópicos e os respectivos *subscribers* de cada tópico. Apesar de auxiliar o *middleware master* em sua tarefa, este pode ser desacoplado, sendo possível a sua disposição em uma réplica diferente do *middleware master*, opção essa escolhida para a realização dos testes.

O *Provider* foi desenvolvido usando o *Java Message Service* em sua versão 1.1, disponível junto ao servidor Glassfish 4.0[43]. Vale ressaltar que atualmente está disponível a versão 2.0 da especificação JMS, lançada em 21 de maio de 2013. No entanto, não há implementação estável desta versão. Por isso, optou-se por utilizar a versão 1.1.

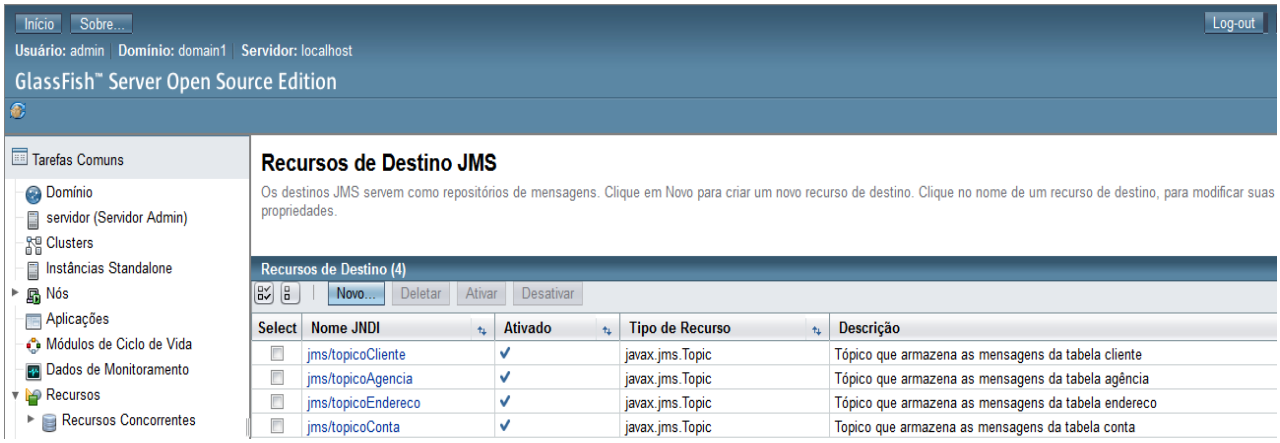
A principal vantagem da utilização do JMS está na confiabilidade do serviço, o qual garante a entrega das mensagens uma e apenas uma vez (SUN, 2002). A entrega é garantida mesmo que o *Provider* falhe ou que ele seja desligado.

Para armazenar os dados de cada uma das tabelas, foram criados quatro tópicos distintos, por meio do console de administração do Glassfish, conforme mostra a Figura 5.1.

No console de administração é possível, ainda, iniciar o serviço de mensagens e realizar quaisquer outras configurações do mesmo.

[42] https://edelivery.oracle.com/linux. Acesso: 28/10/2013.
[43] https://glassfish.java.net/. Acesso: 29/10/2013.

Figura 5.1 - Console de administração do Glassfish, que mostra os tópicos criados para o armazenamento das mensagens.



Fonte: A Autora.

O *Provider* foi disposto em uma máquina física com 300GB de HD, 4GB de memória RAM, Windows 7 Home Basic Service Pack 1 e processador Intel Core i5.

A seção seguinte descreve os elementos necessários para a utilização do sistema *Knowing* nas réplicas da Camada 1.

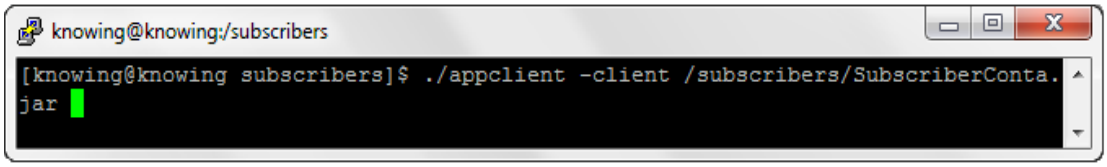
5.1.2. Réplicas da Camada 1

Para o funcionamento do *Knowing* nas réplicas da Camada 1, é necessário, além do SGBD Oracle, que o Glassfish Server, que dá suporte ao sistema de mensagens utilizado, esteja instalado. É preciso também realizar a instalação do Java JDK (*Java Development Kit*), que contém todos os requisitos necessários para a execução de aplicações Java[44].

Para executar o *Knowing*, faz-se uso do utilitário appclient, do Glassfish, conforme mostra a Figura 5.2:

[44] http://www.oracle.com/technetwork/pt/java/javase/downloads/index.html. Acesso: 28/12/2013.

Figura 5.2 - Linha de código utilizada para executar o middleware da Camada 1



Fonte: A Autora.

As duas réplicas da Camada 1 foram configuradas em máquinas virtuais com sistema operacional Oracle Enterprise Linux, HD de 70GB, 2GB de memória RAM e 4GB de *swap.*

Vale ressaltar que uma das réplicas foi colocada nos servidores do Centro de Informática da Universidade Federal de Pernambuco, a qual é acessada por meio de *Virtual Private Network* (VPN - em português, Rede Virtual Privada); e a outra acessada por meio de rede local.

A seguir, serão explanadas em detalhes as tarefas executadas para a realização dos testes no protótipo do modelo *Knowing*. Convém informar que, em razão da simplicidade do minimundo descrito e para fim de testes, todas as tabelas foram definidas de modo a manter a consistência dos dados.

Em um ambiente real, as tabelas deveriam ser analisadas de acordo com as regras de negócio da aplicação a fim de verificar a necessidade da garantia de consistência forte ou eventual.

5.2. Consultas

Os testes realizados com consultas são utilizados para verificação da consistência dos dados. De modo que, enquanto um grupo de escritas era realizado no nó *master*, uma consulta era executada simultaneamente no nó *master* e nas réplicas da Camada 1 a fim de comparar seus resultados. No total, o banco de dados continha 1.259 tuplas.

Foram selecionadas 5 (cinco) consultas, cujos resultados retornados encontram-se no Apêndice A. Os resultados dos experimentos realizados com cada uma das consultas são mostrados a seguir.

5.2.1. Consulta 1

A Consulta 1 retorna a soma dos saldos das contas de todos os clientes de um banco cujos identificadores das contas são maiores que 150. A Figura 5.3 mostra o código SQL da consulta realizada nos três nós durante a execução de um conjunto de inserções na referida tabela.

Figura 5.3 - Consulta 1 realizada para avaliar a consistência dos dados.

```
SELECT SUM ( CONTA.SALDO)  SALDO
  FROM TESTE.TB_CONTA CONTA
 WHERE CONTA.ID_CONTA > 150
```

Fonte: A Autora.

A consulta envolvia apenas uma tabela. A consistência dos dados foi preservada em todos os nós na execução da Consulta 1, conforme os resultados mostrados no Quadro 5.1.

Quadro 5.1- Resultado da execução da Consulta 1 em cada um dos nós da rede

	NO1 (master)	NO2 (remoto)	NO3 (local)
Resultado da consulta	109750	109750	109750

Fonte: A Autora.

5.2.2. Consulta 2

A segunda consulta, conforme mostra a Figura 5.4, tem uma complexidade maior e foi realizada durante a execução de transações de inserção e alteração simultaneamente nas 4 (quatro) tabelas do protótipo. Ela retorna os dados das contas e dos clientes de um determinado grupo de endereços.

Figura 5.4 - Consulta 2 realizada para avaliar a consistência dos dados.

```
SELECT CONTA.ID_CONTA ID_CONTA,
       CONTA.NUM_CONTA NUM_CONTA,
       CONTA.SALDO SALDO,
       CLI.CPF CPF,
       CLI.NOME NOME,
       ENDERECO.ID_ENDERECO END_CLIENTE,
       ENDERECO.CIDADE CIDADE_CLIENTE,
       ENDERECO.ESTADO ESTADO_CLIENTE,
       ENDERECO.ESTADO ESTADO_AGENCIA
 FROM TESTE.TB_CONTA conta
 JOIN TESTE.TB_CLIENTE cli
   ON CONTA.ID_CLIENTE = CLI.ID_CLIENTE
 JOIN TESTE.TB_AGENCIA ag
   ON AG.ID_AGENCIA = CONTA.ID_AGENCIA
 JOIN TESTE.TB_ENDERECO endereco
   ON ENDERECO.ID_ENDERECO = CLI.ID_ENDERECO
WHERE CLI.ID_ENDERECO in (249, 256, 286)
```

Fonte: A Autora.

A consulta executada no nó *master* retornou um total de 144 campos (ver Apêndice A). Nas réplicas da Camada 1, a quantidade de campos cuja consistência foi garantida foi bastante satisfatório, uma vez que apenas um campo da réplica remota foi divergente do nó *master*. Isto é, a taxa de erro[45] foi de 0,69% no nó 2 (remoto) e não foram registrados erros no nó 3 (local), conforme mostrado no Quadro 5.2.

Quadro 5.2 - Resultado da execução da Consulta 2 em cada um dos nós da rede

	NO1 (*master*)	NO2 (remoto)	NO3 (local)
Quantidade de campos retornados	144	143	144
Taxa de Erro	N/A	0,69%	0%

Fonte: A Autora.

[45] A taxa de erro refere-se à quantidade de campos retornados incorretamente (ou não retornados) pela consulta no nó remoto ou local dividido pela quantidade de campos retornados pela consulta no nó *master*. O resultado da divisão é multiplicado por 100.

5.2.3. Consulta 3

A terceira consulta, mostrada na Figura 5.5, também foi realizada durante a execução de transações de inserção, alteração e remoção. A consulta retorna a soma dos saldos de todas as contas do cliente, agrupadas pelo endereço do cliente e nome da agência.

Figura 5.5 - Consulta 3 realizada para avaliar a consistência dos dados.

```
SELECT CLI.CPF CPF,
       SUM (CONTA.SALDO) SALDO,
       CLI.NOME NOME,
       ENDERECO.ID_ENDERECO END_CLIENTE,
       ENDERECO.CIDADE CIDADE_CLIENTE,
       AG.NOME NOME_AGENCIA
 FROM TESTE.TB_CONTA conta
 JOIN TESTE.TB_CLIENTE cli
   ON CONTA.ID_CLIENTE = CLI.ID_CLIENTE
 JOIN TESTE.TB_AGENCIA ag
   ON AG.ID_AGENCIA = CONTA.ID_AGENCIA
 JOIN TESTE.TB_ENDERECO endereco
   ON ENDERECO.ID_ENDERECO = CLI.ID_ENDERECO
   GROUP BY
       CLI.CPF,
       CLI.NOME,
       ENDERECO.ID_ENDERECO,
       ENDERECO.CIDADE,
       AG.NOME
```

Fonte: A Autora.

Com relação à execução da Consulta 3, a mesma retornou 114 campos (ver Apêndice A). Nas réplicas da Camada 1, os resultados também podem ser ditos satisfatórios, uma vez que na réplica remota o resultado foi divergente por apenas uma tupla, que não foi recuperada pela consulta no nó 2 (remoto). No nó 3 (local), mais uma vez, não houve divergência. Ou seja, apenas 5,3% dos campos do nó remoto não retornaram dados consistentes. O Quadro 5.3 mostra a síntese destes resultados.

Quadro 5.3 - Resultado da execução da Consulta 3 em cada um dos nós da rede

	NO1 (*master*)	NO2 (remoto)	NO3 (local)
Quantidade de campos retornados	114	108	114
Taxa de Erro	N/A	5,3%	0%

Fonte: A Autora.

5.2.4. Consulta 4

A consulta de número 4 (ver Figura 5.6) também foi executada em meio a transações de inserção, alteração e remoção. A consulta tem por objetivo retornar a quantidade de clientes, agrupados pelo nome e endereço da agência.

Figura 5.6 - Consulta 4 realizada para avaliar a consistência dos dados.

```
SELECT COUNT (CLI.ID_CLIENTE) QTD_CLIENTES,
       AG.NOME NOME_AG,
       AG.ID_ENDERECO ID_END_AGENCIA
 FROM TESTE.TB_CONTA conta
  JOIN TESTE.TB_CLIENTE cli
    ON CONTA.ID_CLIENTE = CLI.ID_CLIENTE
  JOIN TESTE.TB_AGENCIA ag
    ON AG.ID_AGENCIA = CONTA.ID_AGENCIA
  JOIN TESTE.TB_ENDERECO endereco
    ON ENDERECO.ID_ENDERECO = CLI.ID_ENDERECO
    GROUP BY AG.NOME , AG.ID_ENDERECO
```

Fonte: A Autora.

A consulta executada no nó *master* retornou 33 campos. Nas réplicas da Camada 1, os resultados seguiram conforme esperado: no nó 2 (remoto) apenas 2 campos estavam com dados inconsistentes e não houve inconsistências no nó 3 (local). Desse modo, apenas 6,1% dos campos estavam inconsistentes no nó remoto, conforme mostrado no Quadro 5.4.

Quadro 5.4 - Resultado da execução da Consulta 4 em cada um dos nós da rede

	NO1 (*master*)	NO2 (remoto)	NO3 (local)
Quantidade de campos retornados	33	31	33
Taxa de Erro	N/A	6,1%	0%

Fonte: A Autora.

5.2.5. Consulta 5

A última consulta realizada retorna os dados dos clientes cuja soma do saldo em uma agência seja maior que 1000, como mostra a Figura 5.7:

Figura 5.7 - Consulta 5 realizada para avaliar a consistência dos dados.

```
SELECT CLI.ID_CLIENTE CLIENTE,
       CLI.NOME NOME,
       SUM (CONTA.SALDO) SALDO,
       AG.NOME NOME
  FROM TESTE.TB_CONTA conta
  JOIN TESTE.TB_CLIENTE cli
    ON CONTA.ID_CLIENTE = CLI.ID_CLIENTE
  JOIN TESTE.TB_AGENCIA ag
    ON AG.ID_AGENCIA = CONTA.ID_AGENCIA
  JOIN TESTE.TB_ENDERECO endereco
    ON ENDERECO.ID_ENDERECO = CLI.ID_ENDERECO
GROUP BY  CLI.ID_CLIENTE , AG.NOME, CLI.NOME
HAVING SUM (CONTA.SALDO) >1000
```

Fonte: A Autora.

A Consulta 5 retornou 32 campos (ver Apêndice A). O último teste também foi considerado satisfatório, uma vez que não houve inconsistências em ambos nós, conforme mostra o Quadro 5.5.

Quadro 5.5 - Resultado da execução da Consulta 5 em cada um dos nós da rede

	NO1 (*master*)	NO2 (remoto)	NO3 (local)
Quantidade de campos retornados	32	32	32
Taxa de Erro	N/A	0%	0%

Fonte: A Autora.

5.3. Conclusões

O protótipo que implementa o modelo *Knowing* passou por uma intensa bateria de testes sem a presença de falhas de execução.

Em uma visão geral, os testes realizados foram considerados bastante satisfatórios, uma vez que sua taxa de acerto sempre se apresentou superior a 90% na rede remota e igual a 100% na rede local. Ou seja, descartando a latência da rede o protótipo se mostrou ainda mais eficiente e condizente à proposta inicial.

As inconsistências detectadas não permitem invalidar o modelo, uma vez que os testes foram realizados em ambiente controlado e com poucos recursos de *hardware* e rede disponíveis. Ademais, a funcionalidade proposta que previa o bloqueio das réplicas com dados não atualizados ainda não foi implementada. Os testes confirmaram também a confiabilidade do JMS, uma vez que todas as mensagens foram entregues, isto é, não houve perda de mensagens, além da manutenção da ordem FIFO.

Conclui-se, portanto, que o protótipo do sistema *Knowing* atingiu seu objetivo de ajudar a garantir a consistência dos dados e permitir que o usuário tome decisões sobre a consistência de dados, baseando-se escolha do mesmo sobre quais tabelas necessitam de consistência forte ou não.

6. CONSIDERAÇÕES FINAIS

Este trabalho apresentou o *Knowing,* um modelo para ajudar na garantia de consistência em sistemas de banco de dados relacionais em nuvem. Este capítulo tem por objetivo apresentar as considerações finais sobre o referido trabalho, enfatizando suas principais contribuições, as limitações do modelo, além da sugestão de trabalhos futuros.

6.1. Principais Contribuições

A principal contribuição deste trabalho é a abordagem de atualização ansiosa para ajudar na manutenção da consistência adaptada aos bancos de dados relacionais em nuvem. Ademais, o *Knowing* valoriza conhecimento do usuário sobre a aplicação, permitindo que ele escolha quais tabelas necessitam ou não de garantia de consistência de dados. Desse modo, é possível explorar os diferentes níveis de consistência de dados, contribuindo com a proposta de Eric Brewer em (BREWER, 2012) e permitindo a utilização do modelo DBaaS por diferentes tipos de aplicações, quanto à necessidade de garantia de consistência na nuvem.

Outra contribuição deste trabalho é o desenvolvimento de um protótipo que implementa o modelo *Knowing*, o que possibilitou testar o modelo proposto. A utilização do *Java Message Service* para o tratamento de consistência de dados é outro ponto forte deste trabalho. O JMS trouxe confiabilidade ao modelo de mensagens, uma vez que garante sua entrega sem repetições ou falhas.

É possível, ainda, utilizar o protótipo com outros SGBD, alterando-se apenas o enfileiramento das transações no nó *master* (conforme mostrado na Seção 4.7), que deve ser compatível com o sistema de banco de dados utilizado (o *Oracle Advanced Queueing* empregado para este fim, como dito anteriormente, é compatível apenas com o SGBD Oracle).

Por fim, conclui-se que o objetivo estabelecido no início do desenvolvimento do trabalho - propor um modelo que ajude a garantir a consistência de dados em sistemas de banco de dados relacionais em nuvem, no qual o usuário possa tomar decisões sobre a consistência dos dados - foi cumprido.

A próxima seção apresenta as limitações presentes no desenvolvimento do presente trabalho.

6.2. Limitações

Uma das principais limitações deste trabalho encontra-se no ambiente utilizado para o desenvolvimento e testes do protótipo, uma vez que não houve disponibilidade de máquinas com recursos apropriados para tal. Desse modo, é preciso ressaltar que os testes realizados não podem ser generalizados a uma população distinta da aqui apresentada.

Apesar desse ponto, optou-se por realizar os testes com os mecanismos disponíveis, ao invés de simplesmente abandonar a avaliação. Desse modo, os testes foram executados e refletem a condição do protótipo dentro do ambiente disponível, sendo possível realizar uma avaliação específica.

Outro ponto a ser observado é a ausência de um modelo de recuperação de falhas. Se uma das réplicas (*master* ou da Camada 1) falhar, não há um procedimento de recuperação imediata. Este ponto, no entanto, não pertence ao escopo deste trabalho, apesar de ser recomendado. Para minimizar este problema, diversos recursos disponíveis para o aumento da confiabilidade do *Provider* foram utilizados, a saber: controle da garantia da recepção da mensagem por parte do *subscriber*, persistência das mensagens no tópico para que as mensagens não sejam perdidas em caso de falha do *Provider*, uso de *subscribers* duráveis, que permite ao *subscriber* receber a mensagem mesmo que o *publisher* esteja inativo.

Portanto, apesar da existência de limitações, estas foram consideradas a fim de minimizar seu impacto sobre o resultado final do trabalho.

Na seção seguinte, apresentam-se as recomendações de trabalhos futuros.

6.3. Trabalhos Futuros

Como proposta de trabalhos futuros, recomenda-se, principalmente, a realização de testes em ambientes com a mesma quantidade de réplicas, mas

com maior disponibilidade de recursos de *hardware* e rede, a fim de verificar o comportamento do protótipo.

Ademais, conforme anteriormente recomendado, as réplicas não atualizadas devem ficar indisponíveis para leitura até que sejam recuperadas e as atualizações aplicadas. A implementação dessa funcionalidade também poderá afetar positivamente a garantia de consistência de dados.

Outra recomendação é modificar o modelo proposto para que seja possível replicar os dados em uma granularidade menor que uma tabela. Para isso, faz-se necessário reorganizar os tópicos de modo que estes possam abrigar e gerenciar fragmentos (verticais ou horizontais) das tabelas.

Por fim, recomenda-se o aumento da quantidade de réplicas. Apesar do número de réplicas do modelo estar adequado ao mercado atual, conforme mostrado no Referencial Teórico deste trabalho, esse aumento pode ser benéfico principalmente no que se refere à disponibilidade do serviço. Para isso, no entanto, o protótipo deve ser adaptado a fim de descentralizar o nó *master*.

REFERÊNCIAS

ABADI, Daniel J. **Data Management in the Cloud**: Limitations and Opportunities. IEEE Data Engineering Bulletin. Volume 32, Number 1: pages 3-12. 1 March 2009.

ABADI, Daniel. **Consistency tradeoffs in modern distributed database system design**: Cap is only part of the story. Computer, 45(2):37–42, 2012.

AGRAWAL, D.; EL ABBADI, A.; EMEKCI, F.; METWALLY, A. **Database Management as a Service**: Challenges and Opportunities. IEEE 25th International Conference on Data Engineering (ICDE '09), pp.1709-1716, March 29 2009-April 2 2009.

ALZAIN, M.A.; PARDEDE, E. **Using Multi Shares for Ensuring Privacy in Database-as-a-Service.** In: 44th Hawaii International Conference on System Sciences (HICSS), pp.1-9, 4-7 Jan. 2011.

ARASU, A.; EGURO, K.; KAUSHIK, R.; RAMAMURTHY, R.. **Querying encrypted data.** In: IEEE 29th International Conference on Data Engineering (ICDE), pp.1262,1263, 8-12 April 2013.

BELL, David; GRIMSON, Jane. **Distributed Database Systems**. International Computer Science Series. Addison Wesley (February 26, 1992).

BOG, A.; DOMSCHKE, M.; MUELLER, J.; ZEIER, A. **A framework for simulating combined OLTP and OLAP workloads.** In: 16th International Conference on Industrial Engineering and Engineering Management (IE&EM '09), pp.1675-1678, 21-23 Oct. 2009

BREWER, E. **CAP Twelve Years Later**: How the "Rules" Have Changed. Computer, vol.45, no.2, pp.23-29, Feb. 2012.

BUNCH, C.; CHOHAN, Navraj; KRINTZ, C. **Supporting Placement and Data Consistency Strategies Using Hybrid Clouds.** IEEE Aerospace Conference, pp.1-8, 3-10 March 2012.

BUSCHMANN, Frank; HENNEY, Kevlin; SCHMIDT, Douglas C. **Pattern-Oriented Software Architecture**: A Pattern Language for Distributed Computing. Wiley; Volume 4 edition (May 1, 2007)

CHANG, Fay; DEAN, Jeffrey; GHEMAWAT, Sanjay; HSIEH, Wilson C.; WALLACH, Deborah A.; BURROWS, Mike; CHANDRA, Tushar; FIKES, Andrew; GRUBER, Robert E.; **Bigtable**: a distributed storage system for structured data. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06), vol.7. USENIX Association, Berkeley, CA, USA, 15 November 2006.

CHAPPELL, David. **Enterprise Service Bus**: Theory in Practice. O'Reilly Media (June 2004).

CHEUNG, A. K. Y.; JACOBSEN, H.-A. **Load balancing content-based publish/subscribe systems**. ACM Trans. Comput. Syst. 28, 4, Article 9 (December 2010), 55 pages.

CHEUNG, David W. **Security on Cloud Computing, Query Computation and Data Mining on Encrypted Database**. In: IEEE Technology Time Machine Symposium on Technologies Beyond 2020 (TTM), pp.1, 1-3 June 2011.

CHIHOUB, Houssem-Eddine; IBRAHIM, Shadi; ANTONIU, Gabriel; PÉREZ, María S. **Harmony**: Towards Automated Self-Adaptive Consistency in Cloud Storage. In: 2012 IEEE International Conference on Cluster Computing (CLUSTER), pp.293-301, 24-28 Sept. 2012.

CONNOR, A.G.; CHRYSANTHIS, P.K.; LABRINIDIS, A. **Key-key-value stores for efficiently processing graph data in the cloud.** In: IEEE 27th International Conference on Data Engineering Workshops (ICDEW), pp.88-93, 11-16 April 2011.

COOPER, B; BALDESCHWIELER, E.; FONSECA, R.; KISTLER, J.; NARAYAN, P.; NEERDAELS, C.; NEGRIN, T.; RAMAKRISHNAN, R.; SILBERSTEIN, A.; SRIVASTAVA, U.; STATA, R. **Building a cloud for yahoo!.** IEEE Data Eng. Bull.,vol. 32, no. 1, pp. 36-43, 2009.

COOPER, B.F.; RAMAKRISHNAN R.; SRIVASTAVA, U.; SILBERSTEIN, A.; BOHANNON, P.; JACOBSEN, H.A.; PUZ, N.; WEAVER, D.; YERNENI, R.; **PNUTS**: Yahoo!'s Hosted Data Serving Platform. In: Proceedings of the VLDB Endowment, vol.1, no. 2, pp. 1277–1288, 2008.

CURINO, Carlo; JONES, Evan P. C.; POPA, Raluca Ada; MALVIYA, Nirmesh; WU, Eugene; MADDEN, Sam; BALAKRISHNAN, Hari; ZELDOVICH, Nickolai. **Relational Cloud**: A Database-as-a-Service for the Cloud. In: Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR), Pacific Grove, CA, pp.235-240. January 2011.

CURRY, E. **Message-Oriented Middleware**, in Middleware for Communications (org: Q. H. Mahmoud), John Wiley & Sons, Ltd, Chichester, UK: 2005.

DAS, Sudipto; AGRAWAL, Divyakant; ABBADI, Amr El. **ElasTraS**: An Elastic Transactional Data Store in the Cloud. In: USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '09), in conjunction with USENIX Annual Technical Conference '09.

DAS, Sudipto; AGRAWAL, Divyakant; ABBADI, Amr El. **G-Store**: a scalable data store for transactional multi key access in the cloud. In: Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10).

DAS, Sudipto; AGRAWAL, Divyakant; ABBADI, Amr El. **ElasTraS**: An elastic, scalable, and self-managing transactional database for the cloud. ACM Trans. Database Syst. 38, 1, Article 5 (April 2013).

DATE, C. J. **Introdução a sistemas de bancos de dados**. Tradução da 8ª edição americana. Editora Campus, 2004.

DIESTE, Oscar; JURISTO, Natalia; MORENO, Ana. M.; PAZOS, Juan; SIERRA, Almudena, in **Handbook of Software Engineering and Knowledge Engineering** - Vol. 1: Fundamentals (org: S. K. Chang). World Scientific Pub Co Inc; 1st edition. (January 2002).

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de Banco de Dados**. 6ª Edição. Pearson Addison Wesley. São Paulo, 2011.

EUGSTER, Patrick. **Type-based publish/subscribe**: Concepts and experiences. ACM Trans. Program. Lang. Syst. 29, 1, Article 6 (January 2007)

EUGSTER, P. Th.; FELBER, P. A.; GUERRAOUI, R.; KERMARREC, A. **The Many Faces of Publish/Subscribe**. In: ACM Comput. Surv. 35, 2 (June 2003), p. 114-131.

FETAI, I.; SCHULDT, H., **Cost-Based Data Consistency in a Data-as-a-Service Cloud Environment**. In: IEEE 5th International Conference on Cloud Computing (CLOUD), pp.526-533, 24-29 June 2012.

FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. **Cloud Computing and Grid Computing 360-Degree Compared.** In: Grid Computing Environments Workshop (GCE '08), pp.1-10. Chicago, USA. 12-16 Nov. 2008.

GAO, Aiqiang; DIAO, Luhong. **Lazy update propagation for data replication in cloud computing**. In: 5th International Conference on Pervasive Computing and Applications (ICPCA), pp.250-254, 1-3 Dec. 2010.

GIL, A. Carlos. **Como elaborar projeto de pesquisa**. 4ª edição. São Paulo: Atlas, 2002.

GILBERT, S.; LYNCH, N. **Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services**. SIGACT News, vol.33, pp.51–59, 2002.

KAHLMEYER-MERTENS, Roberto S.; FUMANGA, Mario; TOFFANO, Claudia B.; SIQUEIRA, Fabio. **Como elaborar projetos de pesquisa**: Linguagem e método. Rio de Janeiro, Editora FGV, 2007.

HACIGÜMÜS¸ Hakan; IYER, Bala; MEHROTRA, Sharad. **Providing Database as a Service**. In: Proceedings of the 18th International Conference on Data Engineering, pp. 29–38. San Jose, CA, 2002.

(B)HACIGÜMÜŞ, Hakan; IYER, Bala; LI, Che; MEHROTRA, Sharad. **Executing SQL over encrypted data in the database-service-provider model**. In:

Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD '02). ACM, New York, NY, USA, 216-227.

HSIEH, Meng-Ju; CHANG, Chao-Rui; HO, Li-Yung; WU, Jan-Jan; LIU, Pangfeng. **SQLMR**: A Scalable Database Management System for Cloud Computing. In: International Conference on Parallel Processing (ICPP), pp.315-324. Taipei, Taiwan: 13-16 Sept. 2011.

ISLAM, M.A.; VRBSKY, S.V.; HOQUE, M.A., **Performance analysis of a tree-based consistency approach for cloud databases.** In: International Conference on Computing, Networking and Communications (ICNC), pp.39-44, Jan. 30 2012-Feb. 2 2012.

KAHLMEYER-MERTENS, Roberto S.; FUMANGA, Mario; TOFFANO, Claudia B.; SIQUEIRA, Fabio. **Como elaborar projetos de pesquisa**: Linguagem e método. Rio de Janeiro, Editora FGV, 2007.

KRASKA, Tim; HENTSCHEL, Martin; ALONSO, Gustavo; KOSSMANN, Donald. **Consistency Rationing in the Cloud**: Pay only when it Matters. Proc. VLDB Endow, vol.2, no.1, pp. 253-264 (August 2009).

LENKALA, S.R.; SHETTY, S.; XIONG, Kaiqi. **Security Risk Assessment of Cloud Carrier**. In: 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp.442,449, 13-16 May 2013.

LEVANDOSKI, J. J.; LOMET, D; MOKBEL, M. F; ZHAO, K. K. **Deuteronomy**: Transaction Support for Cloud Data. In: 5th Biennial Conf. on Innovative Data Systems Research(CIDR), pages 123--133, Asilomar, CA, USA, January 2011.

LIN, Kawuu W..; CHEN, Pei-Ling; CHANG, Weng-Long. **A novel frequent pattern mining algorithm for very large databases in cloud computing environments**. In: IEEE International Conference on Granular Computing (GrC), pp. 399-403. Kaohsiung: 8-10 November 2011.

MALHOTRA, N. K. **Pesquisa de marketing**: uma orientação aplicada. 4ª ed., Bookman, 2006.

MARCONI, M. de Andrade; LAKATOS, Eva Maria. **Metodologia Científica**. 3ª Edição. Atlas, 2000.

ORACLE. **Database as a Service**: Reference Architecture – An Overview. An Oracle White Paper on Enterprise Architecture. September 2011. Disponível em: < http://www.oracle.com/technetwork/topics/entarch/oes-refarch-dbaas-508111.pdf >. Acesso: 24/10/2012.

ÖZSU, M. Tamer, VALDURIEZ, Patrick. **Principles of Distributed Database Systems**. Springer; 3rd ed. 2011 edition (March 2, 2011).

PAVLOU, Kyriacos E.; SNODGRASS, Richard T. **Achieving Database Information Accountability in the Cloud.** In: IEEE 28th International Conference on Data Engineering Workshops (ICDEW), pp.147-150. Washington, 1-5 April 2012.

PINHEIRO, J. M. S. **Da Iniciação Científica ao TCC**: Uma Abordagem para os Cursos de Tecnologia.1ª edição. Ciência Moderna, 2010.

RAHIMI, Saeed K.; HAUG, Frank S.. **Distributed Database Management Systems**: A Practical Approach. Wiley-IEEE Computer Society Pr; 1 edition (August 2, 2010).

RAMAKRISHNAN, R.; **CAP and Cloud Data Management**. Computer, vol.45, no.2, pp.43,49, Feb. 2012.

RAMANATHAN, Shalini; GOEL, Savita; ALAGUMALAI, Subramanian. **Comparison of Cloud Database**: Amazon's SimpleDB and Google's Bigtable. In: International Conference on Recent Trends in Information Systems (ReTIS), pp.165-168, 21-23 Dec. 2011.

RAY, Chhanda. **Distributed Database Systems**. Pearson Education India, 2009

SAKR, Sherif; LIU, Anna. **SLA-Based and Consumer-Centric Dynamic Provisioning for Cloud Databases**. In: IEEE Fifth International Conference on Cloud Computing (CLOUD), pp.360–367. Honolulu: 24-29 June 2012.

SAKR, Sherif; ZHAO, Liang; WADA, Hiroshi; LIU, Anna. **CloudDB AutoAdmin**: Towards a Truly Elastic Cloud-Based Data Store. In: IEEE International Conference on Web Services (ICWS), pp.732-733. Washington, DC: 4-9 July 2011.

SATHYA, S. Siva; KUPPUSWAMI, S.; RAGUPATHI, R. **Replication Strategies for Data Grids**. In: International Conference on Advanced Computing and Communications (ADCOM 2006), pp.123-128, 20-23 Dec. 2006

SHIRAZI, Mahdi Negahi; KUAN, Ho Chin; DOLATABADI, Hossein. **Design Patterns to Enable Data Portability between Clouds' Databases**. In: 12th International Conference on Computational Science and Its Applications (ICCSA), pp.117-120. Salvador, 18-21 June 2012.

SOUSA, F. R. C.; MOREIRA, L. O.; MACÊDO, J. A. F.; MACHADO, J. C.. **Gerenciamento de Dados em Nuvem**: Conceitos, Sistemas e Desafios. In: PEREIRA, A. C. M.; PAPPA, G. L.; WINCKLER, M.; GOMES, R. L.. (Org.). Tópicos em Sistemas Colaborativos, Interativos, Multimídia, Web e Bancos de Dados, vol.1, pp. 101-130. Belo Horizonte: SBC, 2010.

STIPIC, Arian; BRONZIN, Tomislav. **How Cloud Computing Is (not) Changing the Way We do BI**. In: Proceedings of the 35th International Convention MIPRO, pp.1574–1582. Opatija: 21-25 May 2012.

SULEIMAN, Basem; SAKR, Sherif; JEFFERY, Ross; LIU, Anna. **On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure**. In: Journal of Internet Services and Applications, vol.3, no.2, pp. 173-193. September 2012.

SUN. **Java Message Service Specification**. Versão 1.1. Palo Alto, CA, Abril, 2002. 140 p.

TANENBAUM, Andrew S.; STEEN, Maarten Van. **Distributed Systems**: Principles and Paradigms. Second Edition. Person Prentice Hall. Upper Saddle River, 2006.

TIWARI, R.G.; NAVATHE, S.B.; KULKARNI, G.J. **Towards Transactional Data Management over the Cloud**. 2nd International Symposium on Data, Privacy and E-Commerce (ISDPE), pp.100-106, 13-14 Sept. 2010.

VOGELS, W. **Eventually consistent**. Commun. ACM, vol. 52, pp. 40–44, January 2009.

WANG, Ximei; YANG, Shoubao; WANG, Shuling; NIU, Xianlong; XU, Jing. **An Application-Based Adaptive Replica Consistency for Cloud Storage**. 9th International Conference on Grid and Cooperative Computing (GCC), pp.13-17, 1-5 Nov. 2010

WEIS, Joel; ALVES-FOSS, Jim. **Securing Database as a Service**: Issues and Compromises. IEEE Security & Privacy, vol. 9, no. 6, pp. 49-55, Nov.-Dec. 2011.

WEITZNER, D. J.; ABELSON, H.; BERNERS-LEE, T.; FEIGENBAUM, J.; HENDLER, J.; SUSSMAN, G. J. **Information accountability**. Communications of the ACM, vol. 51, no. 6, pp. 82–87, June 2008.

XIONG, Pengcheng; CHI, Yun; ZHU, Shenghuo; MOON, Hyun Jin; PU, Calton; HACIGÜMÜS, Hakan. **Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment.** In: Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE'11), pp.87-98. Hannover: 11-16 April 2011.

YU, Tao; QIU, Jie; REINWALD, B.; ZHI, Lei; WANG, Qirong; WANG, Ning. **Intelligent Database Placement in Cloud Environment**. In: IEEE 19th International Conference on Web Services (ICWS), pp.544-551. Honolulu: 24-29 June 2012.

ZHAO, Liang; SAKR, Sherif; LIU, Anna. **Application-Managed Replication Controller for Cloud-Hosted Databases**. In: IEEE Fifth International Conference on Cloud Computing (CLOUD), pp. 922-929. Honolulu: 24-29 June 2012.

(B) ZHAO, Liang; SAKR, S.; FEKETE, A.; WADA, H.; LIU, A. **Application-Managed Database Replication on Virtualized Cloud Environments**. In: IEEE

28th International Conference on Data Engineering Workshops (ICDEW), pp.127-134. 1-5 April 2012.

APÊNDICE A – Resultado das Consultas

Este apêndice apresenta o resultado das consultas que foram executadas no nó *master* durante a realização dos experimentos, com exceção da Consulta 1, cujo resultado foi mostrado no capítulo anterior.

A.1. Consulta 2

A execução da Consulta 2 no nó *master* obteve o seguinte resultado (Quadro A1):

Quadro A1- Resultado da execução da Consulta 2 no nó *master*

ID_CONTA	NUM_CONTA	SALDO	CPF	NOME	END_CLIENTE	CIDADE_CLIENTE	ESTADO_CLIENTE	ESTADO_AGENCIA
561	100	500	1111	CLIENTEA	286	CIDADE	ESTADO	ESTADO
563	100	250	1111	CLIENTEA	256	CIDADE	ESTADO	ESTADO
564	100	250	1111	CLIENTEA	256	CIDADE	ESTADO	ESTADO
565	100	250	1111	CLIENTEA	249	CIDADE	ESTADO	ESTADO
571	100	250	1111	CLIENTEA	286	CIDADE	ESTADO	ESTADO
573	100	250	1111	CLIENTEA	256	CIDADE	ESTADO	ESTADO
574	100	250	1111	CLIENTEA	256	CIDADE	ESTADO	ESTADO
575	100	250	1111	CLIENTEA	249	CIDADE	ESTADO	ESTADO
581	100	250	1111	CLIENTEA	286	CIDADE	ESTADO	ESTADO
583	100	250	1111	CLIENTEA	256	CIDADE	ESTADO	ESTADO
584	100	250	1111	CLIENTEA	256	CIDADE	ESTADO	ESTADO
585	100	250	1111	CLIENTEA	249	CIDADE	ESTADO	ESTADO
591	100	250	1111	CLIENTEA	286	CIDADE	ESTADO	ESTADO
593	100	250	1111	CLIENTEA	256	CIDADE	ESTADO	ESTADO
594	100	250	1111	CLIENTEA	256	CIDADE	ESTADO	ESTADO
595	100	250	1111	CLIENTEA	249	CIDADE	ESTADO	ESTADO

Fonte: A Autora.

A.2. Consulta 3

O resultado desta consulta no nó *master* foi o seguinte (Quadro A2):

Quadro A2 - Resultado da execução da Consulta 3 no nó master

CPF	SALDO	NOME	END_CLIENTE	CIDADE_CLIENTE	NOME_AGENCIA
1111	500	CLIENTEA	396	CIDADE	AGENCIA A
1111	25900	CLIENTEA	237	CIDADE	AGENCIA A
1111	1000	CLIENTEA	375	CIDADE	AGENCIA A
1111	25250	CLIENTEA	384	CIDADE	AGENCIA A
1111	25000	CLIENTEA	325	CIDADE	AGENCIA A
1111	24750	CLIENTEA	381	CIDADE	AGENCIA A
1111	14000	CLIENTEA	226	CIDADE	AGENCIA A
1111	1000	CLIENTEA	383	CIDADE	AGENCIA A
1111	500	CLIENTEA	395	CIDADE	AGENCIA A
1111	1000	CLIENTEA	287	CIDADE	AGENCIA A
1111	1000	CLIENTEA	392	CIDADE	AGENCIA A
1111	500	CLIENTEA	252	CIDADE	AGENCIA A
1111	25000	CLIENTEA	399	CIDADE	AGENCIA A
1111	500	CLIENTEA	399	CIDADE	AGENCIAB
1111	1000	CLIENTEA	344	CIDADE	AGENCIA A
1111	1250	CLIENTEA	286	CIDADE	AGENCIA A
1111	1000	CLIENTEA	342	CIDADE	AGENCIA A
1111	1750	CLIENTEA	256	CIDADE	AGENCIA A
1111	1000	CLIENTEA	249	CIDADE	AGENCIA A

Fonte: A Autora.

A.3. Consulta 4

O resultado da Consulta 4 no nó *master* é mostrado no Quadro A3:

Quadro A3 - Resultado da execução da Consulta 4 no nó *master*

Quantidade de clientes	Nome Agencia	ID Endereço Agencia
4	AGENCIA A	202
23	AGENCIA A	203
3	AGENCIA A	205
4	AGENCIA A	207
2	AGENCIA A	208
101	AGENCIA A	210
100	AGENCIA A	225
100	AGENCIA A	233
100	AGENCIA A	319
76	AGENCIA A	356
94	AGENCIA A	399

Fonte: A Autora.

A.4. Consulta 5

O resultado da Consulta 5 no nó *master* é mostrado no Quadro A4:

Quadro A4 - Resultado da execução da Consulta 5 no nó *master*

ID_CLIENTE	NOME	SALDO	AGENCIA
233	CLIENTEA	25900	AGENCIA A
282	CLIENTEA	1250	AGENCIA A
395	CLIENTEA	25500	AGENCIA A
380	CLIENTEA	25250	AGENCIA A
321	CLIENTEA	25000	AGENCIA A
222	CLIENTEA	14000	AGENCIA A
252	CLIENTEA	1750	AGENCIA A
377	CLIENTEA	25000	AGENCIA A

Fonte: A Autora.