Pós-Graduação em Ciência da Computação

RIVALDO GUIMARÃES DE LIMA JUNIOR

USANDO UM SERVIÇO DE MENSAGERIA CRIADO COM A API JMS PARA MANTER A CONSISTÊNCIA DE DADOS EM BANCO DE DADOS RELACIONAIS DISTRIBUÍDOS

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

RECIFE
2017

RIVALDO GUIMARÃES DE LIMA JUNIOR

USANDO UM SERVIÇO DE MENSAGERIA CRIADO COM A API JMS PARA MANTER A CONSISTÊNCIA DE DADOS EM BANCO DE DADOS RELACIONAIS DISTRIBUÍDOS

Dissertação apresentada à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial à obtenção de grau de Mestre em Ciência da Computação.

ORIENTADOR: **Prof. Dr. Fernando da Fonseca de Souza**

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

L732u Lima Júnior, Rivaldo Guimarães de
Usando um serviço de mensageria criado com a API JMS para manter a consistência de dados em banco de dados relacionais distribuídos / Rivaldo Guimarães de Lima Júnior. – 2017.
99 f.:il., fig.

Orientador: Rivaldo Guimarães de Lima Júnior.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências.

1. Banco de dados. 2. Consistência de dados. I. Lima Júnior, Rivaldo Guimarães de (orientador). II. Título.

025.04 CDD (23. ed.) UFPE- MEI 2017-146

RIVALDO GUIMARÃES DE LIMA JUNIOR

USANDO UM SERVIÇO DE MENSAGERIA CRIADO COM A API JMS PARA MANTER A CONSISTÊNCIA DE DADOS EM BANCO DE DADOS RELACIONAIS DISTRIBUÍDOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre Profissional em 12 de maio de 2017.

Aprovado em: 12/ 05/ 2017

BANCA EXAMINADORA

_____

Prof.Fernando da Fonseca de Souza
Centro de Informática / UFPE
(Orientador)

_____

Prof.Sergio Lifschitz
Pontifícia Universidade Católica dp Rio de Janeiro

_____

Profª. AidaAraujo Ferreira
Instituto Federal de Pernambuco

Dedico às pessoas que, de alguma forma, contribuíram para sua conclusão. Em especial à minha esposa *Joeliza Barreto* que, com muita paciência me deu o suporte necessário para conseguir concluir com êxito esta pesquisa.

AGRADECIMENTOS

Agradeço a Deus pelas bênçãos recebidas, à família pela paciência e dedicação dispensada, aos amigos de trabalho pela ajuda na validação do protótipo em especial a Maurício pela ajuda na construção do ambiente de testes.

RESUMO

De acordo com o Teorema CAP (Consistency, Availability e Partition Tolerance), não é possível alcançar, em um sistema distribuído, e ao mesmo tempo, as características de Consistência, Disponibilidade e Tolerância à Partição (em caso de falhas). Por conta dessa idéia, diversos sistemas de banco de dados em nuvem optaram por relaxar a consistência, priorizando a disponibilidade do serviço. Essa decisão apóia-se no fato de que para muitas aplicações inconsistência nos dados é aceitável quando se consegue a disponibilidade das mesmas, porém, para um número grande de aplicações, essa inconsistência pode representar prejuízos imensuráveis, como no caso de uma aplicação bancária consultando dados inconsistentes de seus clientes. Diante desse quadro se faz necessário investigar e explorar novas opções, que permitam a consistência dos dados, quando estes estão replicados e disponibilizados em nuvem, sem abrir mão da disponibilidade do serviço. Diante do exposto e considerando a Universidade Federal de Pernambuco (UFPE) interessada em replicar seus dados, formando uma nuvem privada, garantindo sua consistência, este trabalho propõe um modelo para manutenção de consistência de dados relacionais replicados, baseado na troca de mensagens entre a aplicação e as réplicas do banco de dados. Realizados testes de cadastros, alterações e exclusões no modelo proposto, foi verificado que o mesmo foi eficaz em manter três réplicas de um banco de dados consistentes entre si.

Palavras-Chave: Banco de Dados como Serviço. Manutenção de Consistência de Dados. Teorema CAP

ABSTRACT

According to the CAP Theorem (Consistency, Availability e Partition Tolerance), it cannot be achieved in a distributed system, and at the same time, the characteristics of Consistency, Availability and Partition tolerance (in case of failures). Considering this idea, various cloud database systems have chosen to relax the consistency, prioritizing service availability. That decision is supported by the fact that for many applications data inconsistency is acceptable should such data be available. However, for a large number of applications, this inconsistency may cause great losses, as in the case of a banking application querying inconsistent data of their customers. Therefore, it is necessary to investigate and explore new options to allow mantaining data consistency, when such data are replicated and available in the cloud, without giving up service availability. Taking the above into consideration as well as that Federal University of Pernambuco (UFPE) is interested in replicating its data, forming a private cloud, ensuring their consistency, this work proposes a model to mantain consistency of replicated relational data, based on the exchange of messages between the application and the database replicas. After performing tests of registrations, changes and exclusions in the proposed model, it was verified that it was effective in keeping three replicates of a database consistent with each other.

Keywords: Database-as-a-Service. Maintenance of Data Consistency. CAP Theorem

LISTA DE FIGURAS

LISTA DE QUADROS

Sumário

1 INTRODUÇÃO

A constante busca por otimização de recursos de Tecnologia da Informação (TI) fez surgir uma demanda crescente por um serviço descentralizado que atenda às necessidades das organizações na medida em que elas surjam. Para atender essa demanda, surgiu o conceito de computação nas nuvens que é a prestação de serviços baseada na Internet. Chamado de *Cloud Computing* (FOSTER, 2008; DONG 2013) esse serviço oferece uma infraestrutura adequada para a escalabilidade que as grandes corporações precisam.

Nesse paradigma, o uso de banco de dados vem crescendo de forma a se tornar um serviço importante. No entanto, o problema ainda enfrentado por empresas que ofertam esse serviço é a manutenção da consistência dos dados em ambientes distribuídos. De acordo com Gilbert e Lynch (2002) consistência, disponibilidade e tolerância à partição (em caso de falha) são características importantes de sistemas gerenciadores de banco de dados, porém, garantir essas características, simultaneamente, e em caso de falhas, em sistemas distribuídos não é possível. Diante desta constatação, pesquisas têm sido feitas (FREITAS, 2014; ISLAM et. al, 2012) com o objetivo de mitigar esse problema.

Uma vez que usando uma infraestrutura distribuída, com replicação dos dados, que pode ser facilmente ofertada na *nuvem,* a disponibilidade e a tolerância à partição são alcançadas, mesmo em caso de falhas que impossibilitem que algumas réplicas do banco de dados sejam alcançadas, na ordem de N-1 onde N é o número de réplicas, cabe investigar meios de garantir a manutenção da consistência dos dados.

Conforme Brewer (2012), a decisão de garantir a consistência e/ou a disponibilidade deve ser tomada só em caso de partição, haja visto que, em caso normal, sem partição essa decisão é desnecessária.

Alguns provedores têm optado por prover uma consistência parcial, optando por garantir a disponibilidade, na qual os dados não são consistentes o

tempo todo e uma janela de tempo é admitida. Essa abordagem é aceitável para sistemas como as redes sociais, porém para uma aplicação como a de uma instituição financeira, por exemplo, é inadmissível (FREITAS, 2014).

1.1 Motivação

Segundo o teorema CAP (Consistency, Availability e Partition Tolerance) (Consistência, Disponibilidade e Tolerância a Partição - me português) em um sistema de banco de dados distribuído não é possível atingir simultaneamente as três características desejáveis de um banco de dados, a saber: Disponibilidade, Consistência e Tolerância à Partição (em caso de falha) (GILBERT e LYNCH 2002).

De acordo com Oszu e Velduriez (2011), “A consistência de uma transação é simplesmente sua aplicação correta”. Vários sistemas de banco de dados distribuídos em nuvem optaram por relaxar a garantia da consistência, priorizando a disponibilidade do serviço (ZHAO et. al, 2012).

Para algumas aplicações, como as redes sociais, esse relaxamento da consistência é válido, pelo fato que não armazenam dados críticos. No Facebook[1], por exemplo, não causaria problema se uma mudança no *status* de um usuário não for imediatamente visível para todos os contatos do mesmo.

No entanto inconsistência de dados pode causar transtornos inconcebíveis para muitas aplicações, como numa aplicação bancária. Assim sendo, em função do relaxamento da consistência, muitos sistemas de Banco de Dados distribuídos em nuvem não estão aptos a receber todos os tipos de aplicações.

O relaxamento da consistência não é o único caminho possível a ser adotado. Em alguns casos, a disponibilidade, que pode ser medida de 0 a 100 por

[1] www.facebook.com

cento, pode ser relaxada a níveis mais baixos, quando não se pode abrir mão da consistência (BREWER, 2012).

A impossibilidade de garantir as três características desejáveis de um banco de dados, quando este está distribuído, fez com que alguns projetistas relaxassem, indiscriminadamente, a consistência dos dados em prol da manutenção de sistemas altamente disponíveis (BREWER, 2012). Isto faz com que esses sistemas não sejam interessantes para algumas aplicações que tenham na consistência dos dados uma necessidade primária.

Considerando tudo isto será apresentada, a seguir, a pergunta de pesquisa do presente trabalho que trata exatamente da manutenção da consistência dos dados em Banco de Dados Distribuídos.

1.2 Problema de Pesquisa

Conforme exposto, percebe-se a necessidade de explorar os diferentes níveis de consistência de um Banco de Dados distribuído, em detrimento ao seu relaxamento.

Desta forma o problema de pesquisa é: ***Como prover a manutenção da consistência dos dados armazenados em banco de dados relacionais distribuídos*****?**

1.3 Objetivos

Para responder à pergunta do problema de pesquisa, os seguintes objetivos foram especificados, divididos em geral e específicos.

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é prover uma solução aplicável ao problema motivador desta pesquisa, através da construção de um Middleware Orientado a Mensagem (MOM), que garanta aplicação de comandos de *delete, insert e update*, assincronamente e simultaneamente, nas réplicas do banco de dados.

1.3.2 Objetivos Específicos

Com o intuito de alcançar o objetivo geral, será necessário alcançar alguns objetivos específicos:

- Definir um modelo de *middleware orientado a mensagem* (MOM);
- Selecionar um servidor que ofereça o serviço de MOM;
- Desenvolver um protótipo para implementar o modelo definido; e
- Experimentar o protótipo para avaliar sua eficácia.

1.4 Justificativa

Na sua obra intitulada: “CAP Twelve Years Later: How the ‘Rules’ Have Changed”, Brewer (2012) faz uma crítica à forma, excessivamente simplificada, como projetistas passaram a encarar os problemas levantados pelo teorema CAP. O referido autor afirma ainda que os citados projetistas passaram a escolher, indiscriminadamente, duas das propriedades desejáveis de um banco de dados, (como, por exemplo, tolerância à partição e disponibilidade; ou consistência e disponibilidade apenas) recomendando, ainda, que a consistência não seja cegamente sacrificada em prol de sistemas altamente disponíveis.

Semelhantemente, Abadi (2012) afirma que o teorema CAP foi mal interpretado, se tornando uma das maiores influências para o surgimento de sistemas com consistência de dados reduzida nos anos seguintes à sua prova.

Diante deste fato fica clara a necessidade de se investigar novas formas de encarar o problema exposto pelo teorema em questão. Simplesmente preparar-se para a partição fazendo a escolha de qual propriedade deve ser mantida não parece suficiente para muitas aplicações que poderiam estar com seus dados distribuídos. Portanto, propor uma alternativa para permitir a manutenção da consistência dos dados e a alta disponibilidade, em sistemas hospedados na nuvem, mesmo em caso de partições (falhas que impossibilitem acesso a uma ou mais réplicas do banco de dados) é um importante trabalho de pesquisa para a área.

1.5 Contribuições da pesquisa

Espera-se alcançar as seguintes contribuições ao final desta pesquisa:

- Apresentação de um estudo dos temas de Banco de dados como Serviço, e dos problemas recorrentes quanto à manutenção da consistência de dados nesse ambiente;
- Um modelo de middleware orientado a mensagem (MOM) para desenvolver um mecanismo de manutenção da consistência em caso de partição;
- Utilização de uma abordagem de replicação descentralizada de dados e propagação dos mesmos; e
- Desenvolvimento de um protótipo que permita a manutenção da consistência dos dados entre as réplicas dos bancos de dados.

1.6 Metodologia

De acordo com Kahlmeyer-Mertens et. al. (2007), "O método cientifico é entendido como o conjunto de processos orientados por uma habilidade crítica e criadora voltada para a descoberta da verdade e para a construção da ciência hoje". A seguir é apresentada a classificação desta pesquisa.

1.6.1 Natureza da Pesquisa

Esta pesquisa classifica-se como aplicada por visar gerar conhecimentos práticos, dirigidos à solução de problema específico.

1.6.2 Finalidade

Quanto à finalidade, esta pesquisa classifica-se como exploratória e explicativa. A pesquisa exploratória tem o objetivo de conhecer o tema de estudo inicialmente. Neste trabalho será realizada por meio da revisão da literatura. A pesquisa explicativa tem o objetivo de identificar os fatores que fazem parte do fenômeno. Para alcançar esse fim será usado método experimental.

1.6.3 Meios

Quanto aos meios, esta pesquisa classifica-se como experimental, seguindo a idéia de Pinheiro (2010) que afirma que a pesquisa experimental depende da determinação de um objeto de estudo, identificando as variáveis que podem influenciá-lo e observando os efeitos que as variáveis produzem no objeto.

1.7 Descrição

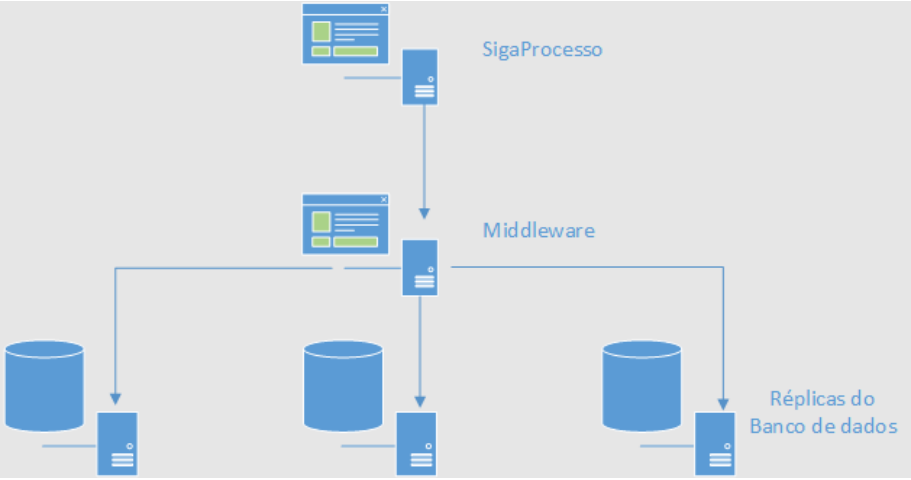
A proposta desta pesquisa é apresentar uma solução aplicável ao problema da manutenção da consistência dos dados em banco de dados distribuídos. Garantir a disponibilidade é o objetivo de uma aplicação hospedada na nuvem, no entanto, essa garantia pode requerer relaxamento de outras, segundo o teorema CAP (Gilbert e Lynch, 2002) a consistência tende a ser relaxada, tornando inapropriado o seu uso para algumas aplicações.

Para alcançar a solução proposta, será criado um protótipo de sistema de mensageria, responsável por receber mensagens de atualização dos dados (*delete*, *insert* e *update*), encaminhá-las às réplicas do banco de dados, permitindo assim, que todas as réplicas recebam as atualizações e aplique-as em suas cópias do banco de dados com uma diferença mínima no *timestemp* de aplicação em cada uma delas, provendo assim a consistência.

Na Figura 1.1 pode-se ver como a solução se comportará: o aplicativo (sigaProcesso) responsável por gerar os comandos de atualizações não aplicará o comando diretamente no banco de dados, mas criará uma mensagem contendo esses comandos e enviará a mesma ao servidor de mensagem (Middleware). Este se encarregará de disponibilizar as mensagens para que os clientes (Réplicas) possam recuperá-las e aplicá-las.

Na Figura 1.1 pode-se ver que como a solução se comportará a fim de prover a consistência entre as réplicas. O *sigaProcesso*, estudo de caso neste trabalho, manterá contato com o *middleware* responsável por gerencias as mensagens do protótipo. O middleware se encarregará de enviar as mensagens aos clientes (réplicas do banco de dados).

Figura 1.1 - Protótipo proposto



Fonte: Elaborada pelo Autor (2017).

sigaProcesso – Máquina responsável por gerar atualizações. É a maquina que hospeda a aplicação que gera as atualizações a serem aplicadas às réplicas do banco de dados. Nesta máquina está executando a aplicação *sigaProcesso* da Universidade Federal de Pernambuco (UFPE).

Middleware – Máquina que hospedará o *Glassfish4.1*[2], *Provider* responsável por gerenciar as mensagens criadas com o conteúdo vindo de *sigaProcesso*. Nesta host também estarão hospedados os *Middleware Orientados a Mensagens* que criarão e consumirão as mensagens aplicando seu conteúdo (transações de *insert, delete e update*) às réplicas do banco de dados.

Réplicas – Máquinas que hospedarão o sistema gerenciador do banco de dados (SGBD), para esta pesquisa o Oracle 11g Release2, e as cópias do banco de dados.

Contando com as garantias que a API JMS oferece (garantia de entrega das mensagens, garantia da ordem de entrega, garantia de que uma mensagem só é

[2] https://glassfish.java.net/docs/4.0/reference-manual.pdf

entregue uma vez para cada cliente), pode-se implementar um *Middleware Orientado à Mensagem* para a execução acima proposta.

O servidor Web a ser implementado é o Glassfish 4.1, escolhido por sua robustez e simplicidade de gerenciamento, se comparado a alguns concorrentes como o ActiveMQ, JBossMQ, OpenJMS entre outros. O sistema gerenciador de banco de dados será o Oracle 11g versão 2.0.1[3], que é a versão utilizada no *sigaProcesso* da UFPE.

1.8 Estrutura do Trabalho

Neste capitulo, foi apresentado o problema de pesquisa bem como os objetivos gerais e específicos para responder à pergunta de pesquisa. Além disso, foram apresentadas a justificativa e a motivação para realização deste trabalho, descrita a metodologia utilizada e destacadas as contribuições esperadas.

O restante do trabalho está estruturado da seguinte forma:

- Capítulo 2 – Estado da arte: Expõe o referencial teórico, que aborda os temas Database as a Service, Middleware Orientado a Mensagem e Consistência de Dados, além de tratar trabalhos relacionados a este.
- Capítulo 3 – Modelo para manutenção de consistência de dados em banco de dados relacionais distribuído: Apresenta o modelo proposto por meio da explanação de sua arquitetura.
- Capítulo 4 – Protótipo para aplicação do modelo: Descreve os detalhes para implementação do protótipo.
- Capítulo 5 – Experimentos: Descreve as avaliações realizadas bem como expõe a analise dos resultados.

[3] http://docs.oracle.com/cd/E11882_01/index.htm

- Capítulo 6 – Considerações finais: Expõe as conclusões sobre o estudo e a proposta de trabalhos futuros.

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

O próximo capítulo apresenta uma revisão da literatura, a fim de fundamentar o presente trabalho, onde serão abordados os temas de Banco de dados distribuídos, Consistência de dados, Teorema CAP, PACELC e Middleware Orientado a Mensagem.

2 ESTADO DA ARTE

Este capítulo visa abordar detalhes das tecnologias e conceitos empregados na solução proposta bem como trabalhos relacionados.

2.1 Database as a Service - DBaaS

A computação em nuvem (cloud computing) é uma infraestrutura baseada na Internet para oferecer serviços de TI a clientes geograficamente distribuídos. É um paradigma da computação, relativamente recente que cresce a uma velocidade considerável. Várias corporações têm optado por usar os serviços da *nuvem* como opção frente às suas infraestruturas custosas.

Tratando-se de banco de dados, um serviço chamado Banco de dados como um serviço, *Database as a Service* em inglês vem se firmando nesse ambiente (FREITAS, 2014). Com surgimento datado de 2009, com o lançamento do MySQL[4] em nuvem pela Amazon.com[5] (FREITAS, 2014) sua aceitação vem crescendo, à medida que tem se tornado consistente e seguro.

O propósito central do DBaaS é o provimento de um sistema de banco de dados com suas funcionalidades nas nuvens, tirando dos clientes a responsabilidade de mantê-lo.

Por conta da impossibilidade apontada pelo teorema CAP de se alcançar as características ACID em SGBD tradicionais, quando estes estão distribuídos, vêm surgindo SGBD específicos como os NoSQL. Segundo Elsmari(2001), o problema da falta de suporte a particionamento dinâmico de dados limita a escalabilidade e utilização de recursos.

[4] http://dev.mysql.com/doc/refman/5.7/en/

[5] https://aws.amazon.com/pt/blogs/aws/introducing-rds-the-amazon-relational-database-service/

2.1.1 Conceitos

Ainda não há um consenso entre os principais autores a respeito do paradigma *Database as a Service*. A seguir serão relacionadas algumas definições.

De acordo com a Oracle (2016), se trata de uma abordagem operacional e arquitetural que permite que provedores de TI entreguem funcionalidades de banco de dados como um serviço para consumidores.

Isso implica que o DBaaS é um serviço formado por funcionalidades gerais dos SGBD como *backup*, balanceamento de carga, gerenciamento de memória entre outros, oferecidos aos consumidores pelos prestadores do serviço que se responsabilizam em manter o serviço em uso independente de qualquer circunstância adversa, dentro de um acordo previsto.

Para Agrawal et al. (2009), trata-se de um novo paradigma de banco de dados, o qual consiste em um serviço terceirizado, ofertado por um prestador como um serviço, garantindo acesso universal pela Internet.

Isso que dizer que é uma área a ser explorada, mesmo sendo um tema conhecido “Banco de Dados”, novas técnicas e novos conceitos devem ser integrados aos já existentes, para tornar possível a oferta do serviço.

Independentemente da definição, é um serviço muito interessante, por tirar do cliente preocupações como aquisição e manutenção de *hardware* e *software*, representando uma economia considerável e ainda garantindo acessibilidade 24 horas por dia a partir de qualquer lugar.

No entanto, mesmo considerando essas vantagens ao usar esse serviço, cuidados são necessários na hora de contratá-lo. Algumas garantias devem ser exigidas, a saber:

- Garantia de confidencialidade - O cliente precisa ter garantido que seus dados não serão manipulados por terceiros, com ou sem permissão do prestador;

- Garantia de Disponibilidade - Os dados devem estar disponíveis a qualquer momento para clientes em quaisquer lugares; e

- Garantia de consistência - Os dados devem ser consistentes. Independentemente de qual réplica dos dados o cliente esteja acessando, precisa ter certeza de que os dados naquela réplica são corretos.

Outra garantia desejável, mas não imprescindível, é a baixa latência, que vai depender de vários fatores que não estão, necessariamente, sob controle do prestador, como por exemplo: número de usuários conectados simultaneamente e custos das consultas realizadas pelos usuários entre outros. A Figura 2.1 mostra alguns serviços oferecidos em um DBaas.

Figura 2.1 - Serviços Oferecidos por um DbaaS

Fonte: Freitas (2014).

2.1.2 Características

O paradigma DbaaS, fazendo parte da computação em nuvem, tem as características peculiares dela, como baixo custo, escalabilidade, disponibilidade, entre outras.

2.1.3 Replicação de dados

Para oferecer um serviço de banco de dados hospedado em nuvem a replicação dos dados é uma técnica muito usada. Trata-se da criação de cópias dos dados e armazenamento distribuído geograficamente. Para Sathya (2006), a idéia geral é armazenar cópias dos dados em diferentes locais para que os mesmos possam ser acessados, ainda que uma das cópias esteja inacessível. Além disso, dados mantidos mais perto, geograficamente, do interessado por meio da replicação, pode melhorar o desempenho do acesso.

A replicação de dados pode ser classificada em síncrona e assíncrona. Na replicação síncrona os dados replicados apresentam um mesmo estado em qualquer réplica num momento qualquer, ou seja: replicação síncrona garante consistência forte. Na replicação assíncrona réplicas podem, em um momento qualquer, ter estados dos dados diferentes. Isso quer dizer que em algum momento duas ou mais réplicas podem ter valores diferentes para um mesmo item de dados. Essa abordagem garante consistência fraca.

O que determina essa classificação é a forma como as atualizações são propagadas entre as réplicas, optando-se por uma atualização preguiçosa ou ansiosa.

Atualização Ansiosa

Nessa forma as atualizações são realizadas no contexto da transação, isto é, quando uma transação termina, todas as réplicas possuem o mesmo valor para o item de dado envolvido na transação (OSZU e VALDURIEZ, 2011).

Uma das estratégias usadas por esse modelo é o *commit* em duas fases (*two-phase commit* – 2PC) (OSZU e VALDURIEZ, 2011). Essa estratégia divide a execução do *commit* em duas etapas: primeiro o sistema envia um aviso para

preparar o *commit* a todos os servidores (réplicas) envolvidos: depois o sistema envia uma aviso para “realizar” o *commit*, caso tenha recebido todas as respostas positivas, ou para “abortar” o *commit*, caso não receba todas as repostas positivas.

Essa estratégia pode causar problemas de desempenho, pelo fato que, durante a execução da transação, o dado a ser atualizado fica bloqueado para outras transações. Além disso, segundo Ozsu e Valderiez (2011), se uma das réplicas não puder ser atualizada, a transação não será finalizada. Isso demanda busca por alternativas, como a comunicação descentralizada em grupo, proposta neste trabalho.

Atualização Preguiçosa

Nessa forma a atualização é realizada em uma réplica e repassada às demais de modo assíncrono. De acordo com Gao e Diao (2010), transações independentes aplicam as atualizações às réplicas, após confirmação da aplicação das mesmas na réplica original. Deste modo a atualização dos dados é um processo separado da propagação.

- Umas das estratégias de atualização preguiçosa, utilizada por diferentes SGBD tradicionais, é o *store and forward* (armazenar e transmitir). Nessa estratégia, um local primário é atualizado e repassa as atualizações às réplicas que as aplica em lotes. O local primário preserva a consistência determinando a ordem de serialização das transações.

Diminuição do tempo de resposta e capacidade de lidar com maior número de requisições estão entre as vantagens dessa estratégia (GAO e DIAO, 2010).

O Quadro 2.1, mostra um comparativo entre as características das duas estratégias discutidas anteriormente.

Quadro 2.1 – Características das estratégias de atualização ansiosa e preguiçosa

Estratégia	Método	Consistência	Vantagem	Desvantagem
Ansiosa	Atualizações realizadas no contexto da transação	Forte	Garantia da manutenção de uma consistência forte	Alto custo de implementação e aumento da latência das transações.
Preguiçosa	Atualizações aplicadas a uma réplica, repassada às demais assincronamente	Fraca	Tempo reduzido de resposta e capacidade de lidar com um maior número de requisições	Ocorrência de inconsistência temporária

Fonte: Adaptado de Freitas (2014).

2.1.4 Objetivos

De acordo com Sakr (2012), sistemas de banco de dados distribuídos bem sucedidos têm os garantes as seguintes características::

- Disponibilidade - O serviço deve estar sempre acessível, mesmo em caso de falha de rede que torne um host envolvido na prestação do serviço, inalcançável ou até mesmo um *Data Center* inteiro ficar *off-line*;
- Escalabilidade - Deve ser capaz de manter em funcionamento grandes banco de dados com altas taxas de solicitação e baixas taxas de latências. Ainda deve ser capaz de replicar dados e redistribuí-los automaticamente, e balancear a carga de trabalho entre servidores;
- Elasticidade - Deve ser capaz de lidar com as mudanças de necessidades das aplicações; e
- Desempenho - Em ambiente de nuvem o preço é definido pelo que se usa, de modo que aumenta linearmente com o armazenamento, largura da banda de rede e capacidade de computação. Por isso o desempenho tem uma influência direta sobre o preço.

Cooper (2008) define as seguintes garantias, que podem ser tomadas também como objetivos de DBaaS:

- Multi hospedagem - deve ser capaz de hospedar várias aplicações (inquilinos) numa mesma arquitetura de *hardware* e *software*. Esses inquilinos devem compartilhar informação, mas ter seus desempenhos isolados um do outro;
- Carga e balanceamento de hospedagem - Deve ser capaz de mover carga entre servidores para que recursos de *hardware* não fiquem sobrecarregados;
- Segurança - A segurança é crítica, pois uma violação pode ter impacto sobre todas as aplicações hospedadas; e
- Operacionalidade - A operacionalidade dos sistemas em nuvem deve ser fácil, de modo que uma equipe central pode gerenciá-los em grande escala.

Esta pesquisa se enquadra no paradigma DbaaS quando visa oferecer um serviços de banco de dados distribuído com as funcionalidades mínimas dos SGBD tradicionais como segurança, controle de acesso, integridade dos dados, realização de backup entre outras. Ademais oferece uma alternativa à manutenção da consistência dos dados.

2.2 Teorema CAP

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

Tal proposição foi, originalmente, concebida por Brewer (2000) e ficou conhecida como Teorema CAP (Consistency, Availability e Partition Tolerance). Consistência, Disponibilidade e Tolerância a Partição, em Português. Os sistemas de bancos de dados distribuídos se encaixam nesse perfil.

2.2.1 Conceitos

CAP é o teorema que provou ser impossível garantir, simultaneamente e em caso de falhas, as três características desejáveis de um banco de dados, quando este está distribuído.

- Consistência - equivalente a ter uma única cópia de dados;

- Disponibilidade dos dados (para atualizações); e

- Tolerância a partições de rede.

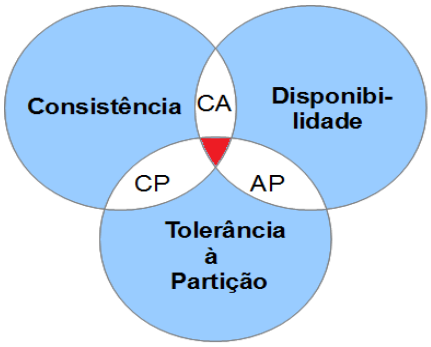
De acordo com Zhao et. al. (2012), consistência significa que ter todos os registros iguais em todas as réplicas em um dado momento, disponibilidade significa que todas as réplicas estão aptas a receber atualizações e a tolerância à partição é ter o sistema em funcionamento mesmo que algumas réplicas tenham problemas de comunicação.

Brewer (2012) afirma que em suas pesquisas percebeu que à medida que a disponibilidade aumenta, diminui a consistência, ou seja, quanto maior a garantia de disponibilidade mais comprometida fica a consistência. Constatação que torna, muitas vezes, o uso de banco de dados replicado impróprio, levando em consideração que algumas aplicações não podem tolerar inconsistência nos dados.

Em resumo, pode-se admitir que, em caso de falha, apenas duas das três características CAP podem ser garantidas. Por exemplo, um sistema poderia garantir disponibilidade e tolerância à partição abrindo mão da consistência se a

mesma pudesse ser relaxada, outro poderia garantir disponibilidade e consistência abrindo mão da tolerância à partição (Vide Figura 2.2).

Figura 2.2 - Representação visual do Teorema CAP



Fonte: Adaptado de CouchDB: The Definitive Guide – Eventual Consistency (2016)[6]

2.2.2 Disponibilidade

A opção por contratar um serviço de banco de dados em nuvem, geralmente, está associada a economia, escalabilidade, bom uso dos recursos e a garantia de disponibilidade próximo dos 100%.

Alcançar esses benefícios em um *Data Center* local é um processo custoso e difícil. Portanto, contratar um serviço desses se torna um caminho interessante para as corporações.

A disponibilidade diz respeito a ter garantias que seus dados estarão disponíveis a qualquer momento de qualquer lugar. Para atender esse requisito as prestadoras do serviço usam estratégias como a replicação de dados, que faz parte da proposta desta dissertação.

[6] <http://guide.couchdb.org/editions/1/en/consistency.html>. Acesso: 05/01/2017.

Existem três estratégias de replicação de dados: replicação parcial, replicação total e ausência de replicação (Elsmari e Navathe, 2011).

Na replicação total o banco de dados é replicado por inteiro em diversas cópias, podendo as mesmas ser dispostas em locais distintos geograficamente.

Na replicação parcial apenas algumas partes do banco de dados são replicadas. Essa estratégia é conveniente quando o projetista quer aproximar os dados de seus interessados. Por exemplo, uma empresa nacional pode optar por manter uma cópia de seus dados de vendas em cada região do país, pois assim ao consultar suas vendas um usuário estaria consultando uma réplica de sua região desses dados, tornando assim mais rápida essa consulta.

Na ausência de replicação, o banco de dados é fragmentado e seus fragmentos armazenados em uma dada localização.

A replicação dos dados proporciona disponibilidade dos mesmos, pois evita partição do serviço porém, pode gerar problemas de consistências nos dados replicados ou de bloqueio do aplicativo cliente à espera da atualização das réplicas.

2.2.3 Tolerância à Partição

Capacidade de um sistema lidar com problemas de comunicação com partes do banco de dados. No caso de uma replicação total dos dados, é a capacidade de direcionar os usuários às cópias ativas, quando alguma cópia estiver indisponível na rede.

2.2.4 Consistência de Dados

A estratégia adotada para garantia da disponibilidade, replicação total ou parcial pode gerar problemas com a consistência dos dados, pelo fato que é impossível garantir que em um dado momento todas as réplicas do banco de dados estarão num mesmo estado, ou seja, todas as réplicas terem exatamente os mesmos dados.

A consistência é uma característica muito importante: o estado correto dos dados mantidos em um sistema em alguns casos é imprescindível.

Para Elsmari e Navathe (2011), uma transação de leitura e ou escrita no banco de dados possui as seguintes características desejáveis:

- **Atomicidade** - Uma transação deve ser executada completamente ou não ser realizada;
- **Preservação da Consistência** - Uma transação deve levar o banco de um estado consistente a outro estado consistente;
- **Isolamento** - Garante que uma transação não sofrerá interferência pela execução de uma outra concorrentemente.
- **Durabilidade** - Todas as mudanças no estado do banco de dados devem ser persistentes, independente de possíveis falhas.

Conhecidas como ACID, essas características garantem um banco de dados consistente e foram alcançadas em bancos de dados relacionais tradicionais. No entanto, se tratando de banco de dados em nuvem, sua arquitetura baseada na replicação dos dados torna a obtenção dessas características mais complicada.

É essa arquitetura de replicação dos dados, presente nos bancos de dados distribuídos, que dificulta a garantia das características ACID (ABADI, 2009). A consistência nesse caso só se dá quando todas as réplicas têm valores idênticos

para cada um de seus elementos (OSZU e VALDURIEZ, 2011). A consistência pode ser ainda classificada em dois grupos: consistência forte e consistência fraca.

- Consistência forte

Nesse modelo de consistência, o usuário tem a certeza que qualquer operação de inserção, atualização ou remoção realizada no banco de dados estará disponível para qualquer outro usuário que consultar esse mesmo dado.

Para Ramakrishnan (2012), a consistência forte significa que quando uma solicitação de gravação retorna com êxito, todas as consultas subsequentes verão os efeitos desta gravação, independentemente de replicação e falha de partição.

Apesar de ser uma garantia desejada por qualquer usuário de banco de dados, deve-se considerar que quanto maior o nível de consistência desejado, maior será o custo para mantê-lo.

- Consistência fraca

Nesse modelo não há garantias que após uma escrita os usuários que consultarem os dados terão sua versão mais recente. Dentre as possibilidades desse modelo a consistência eventual é uma das mais conhecidas.

Esta garante que os dados serão tornados consistentes eventualmente (ISLAM, 2012), isto quer dizer que o banco de dados será atualizado, porém em momento desconhecido.

A escolha entre o tipo de consistência forte ou fraca depende do tipo de aplicação, sendo essa avaliação responsabilidade do projetista.

Algumas aplicações permitem o uso da consistência fraca, como as aplicações Web 2.0, porém outras exigem a leituras de dados consistentes, necessitando de garantia de consistência forte. Pode-se ainda observar que uma

mesma aplicação pode necessitar de consistência forte em alguns dados e fraca em outros.

2.3 PACELC

O PACELC (Abadi, 2012) é uma abordagem mais abrangente do problema enfrentado com sistemas de banco de dados distribuídos. Diferente do teorema CAP, segundo o autor, é uma descrição mais completa das vantagens e desvantagens de consistência para um banco de dados distribuído.

Ainda segundo Abadi (2012), quando há uma partição o sistema deve resolver o conflito entre garantir a disponibilidade ou a consistência; do contrário, sem uma partição, o conflito é entre a consistência e a latência. Afirma ainda que ignorar a latência em prol da consistência é um descuido pois a mesma, latência, está presente em todos os momentos durante a operação do sistema.

Inserindo a questão da latência esse teorema mostra que mesmo em um ambiente robusto, preparado para evitar partições ainda existe um conflito a ser resolvido, pois latência, sendo o tempo entre o início e o término de um evento, diz muito sobre a qualidade de um serviço, ainda mais quando um grande volume de dados precisa ser trafegado pela rede.

2.4 Middleware Orientado a Mensagem (MOM)

Trata-se de um modelo de comunicação entre sistemas de software, em sistemas distribuídos, caracterizado pelo baixo acoplamento dos mesmos, utilizando a troca de mensagens.

Basicamente, permite que aplicações distintas se comuniquem de forma transparente usando um intermediário (serviço de mensageria) que cuida de

receber e encaminhar mensagens. A grande vantagem é que as aplicações envolvidas na comunicação não precisam se conhecer, elas apenas precisam criar, enviar, receber e processar mensagens. A Figura 2.3 ilustra como funciona um MOM.

Cada caixa na Figura 2.3 representa um sistema envolvido na comunicação no sistema de mensageria, a saber:

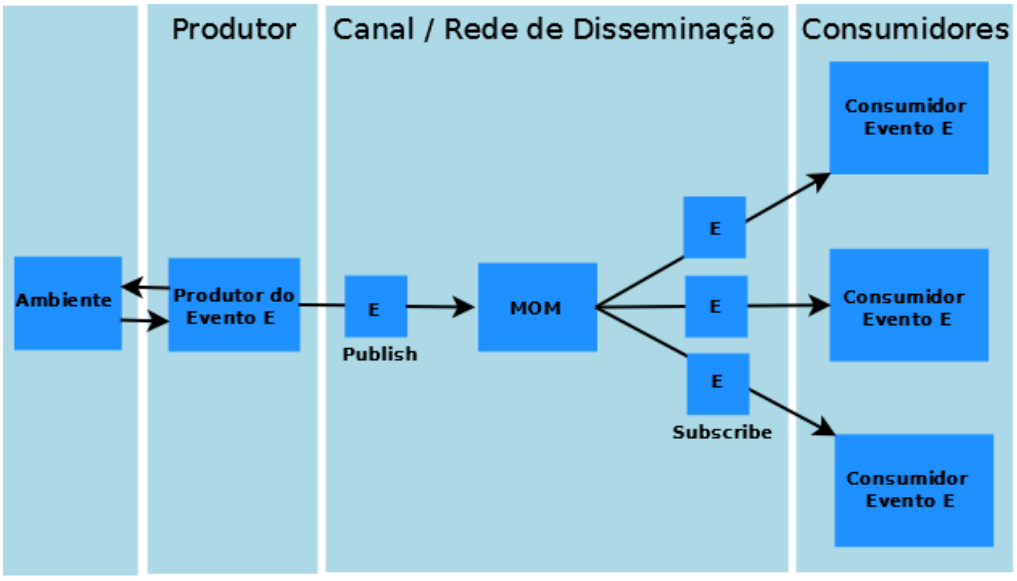
Ambiente: Aplicação responsável por gerar dados que formaram as mensagens.

Produtor: Middleware responsável por gerar uma mensagem e encaminhá-la ao *Provider*.

E: Mensagem criada e enviada ao *Provider* usando o método *publish* e recebida pelos consumidores usando o método *subscribe*.

Consumidor: Middleware responsável por receber as mensagens (clientes inscritos num tópico)

Figura 2.3 - Representação visual do Paradigma MOM



Fonte: Elaborada pelo Autor (2017).

Pode-se verificar, na Figura 2.3, que os agentes envolvidos, produtor e consumidor, não mantêm uma comunicação direta, mas indireta por meio de um provedor (bloco MOM na figura 2.3) isso facilita o trabalho dos programadores, pois para criar uma aplicação produtora de mensagens não é necessário conhecer os consumidores e vice versa.

Ainda sobre a Figura 2.3, a mesma mostra o paradigma publicar e assinar (*publish-subscribe*) no qual um produtor envia mensagens para o provedor e essas podem ser recebidas por mais de um consumidor. No entanto, existe também o paradigma da fila (*queue*) no qual um produtor envia a mensagem que tem um destinatário específico e único.

A respeito dos paradigmas *publish-subscribe* e *queue*, serão abordados com mais detalhes na seção correspondente do JMS.

2.5 Java Message Service (JMS)

JMS é uma especificação para programas Java[7] distribuídos se comunicarem indiretamente, unificando os paradigmas publicar-assinar e fila de mensagens (COULOURIS, 2013).

JMS é uma API Java da SUN que permite que aplicativos criem, enviem, recebam e leia mensagens (SUN, 2002).

É o padrão de Middleware Orientado a Mensagem da Sun para a linguagem JAVA (SUN, 2002), facilitando a interligação de aplicações de forma simples e objetiva. JMS abstrai os conceitos para o desenvolvedor, oferecendo recursos suficientes para dar suporte a aplicações sofisticadas (SUN, 2002).

A principal vantagem de se usar JMS é a garantia da entrega que essa API oferece, possibilitando entregar mensagens mesmo a *hosts* temporariamente

[7] Disponível em: <http://www.oracle.com/br/java/overview/index.html>. Acesso: 15/01/2016

desligados da rede, por meio da permanência da mensagem em um repositório até que a mesma seja entregue a seu destinatário. Vale lembrar que esse tempo de permanência da mensagem no repositório é ajustável, não permanecendo indefinidamente, e que uma mensagem só é entregue uma única vez para um cliente.

Outra vantagem é o uso de domínios, podendo ser o *Point-to-point model* ou *Publish/Subscribe Model* (SUN, 2002).

No domínio *Point-to-point* (ponto-a-ponto em Português) as mensagens são enviadas a filas específicas, sendo retiradas por clientes na ordem em que foram inseridas: primeira a entrar é a primeira a ser retirada (FIFO). Nesse conceito cada mensagem pode ser retirada apenas por um cliente.

No domínio *Publish/Subscribe* (Publicar-Assinar em Português) as mensagens são enviadas a um nó em uma hierarquia de conteúdo. Cada editor pode enviar e/ou receber mensagens e geralmente esse diálogo é anônimo, ou seja, após enviar uma mensagem o criador da mesma não precisa conhecer os destinatários. O mesmo conceito se aplica ao receptor: ao receber uma mensagem, não precisa conhecer quem foi o responsável por seu envio.

Além de o middleware orientado a mensagens ser fracamente acoplado (SUN, 2002), o que facilita a integração de aplicações, pode executar suas tarefas de forma assíncrona, ou seja, o criador das mensagens não fica aguardando que o consumidor as consuma. Mesmo que uma aplicação consumidora não consuma suas mensagens imediatamente, a aplicação criadora continua sua rotina normalmente.

2.5.1 Comunicação Síncrona

É uma forma de integrar aplicações, na qual a aplicação que cria e envia uma mensagem fica esperando uma confirmação de que o(s) cliente(s)

recebeu(ram) a mesma para continuar sua rotina de execução. Essa forma é muito utilizada por sistemas convencionais de comunicação (IBRAHIM, 2012).

Tem sua principal vantagem no fato de que toda operação finalizada deve ter sido realizada completamente. Por exemplo, quando todas as aplicações envolvidas no processo são pertencentes a uma mesma empresa.

Sua desvantagem, em alguns cenários, é que algumas aplicações não podem esperar que suas mensagens sejam recebidas por seus clientes. Por exemplo, um e-commerce não poderá ficar parado esperando confirmar um pagamento de cartão de crédito, impedindo que o cliente possa efetuar outras operações.

2.5.2 Comunicação Assíncrona

Com mensagem assíncrona o baixo acoplamento é mais evidente, pelo fato de que uma aplicação ao criar e enviar uma mensagem não precisa esperar confirmação que essa foi consumida pelo(s) cliente(s).

Em ambientes corporativos, nos quais várias aplicações, criadas por diferentes empresas, consumirão mensagens simultaneamente, essa forma se mostra a mais indicada por facilitar a integração de novas aplicações de forma dinâmica, sem a necessidade de refazer as já integradas.

Mesmo sendo indicada, se tratando de serviços prestados nos quais para o prestador é importante se certificar do recebimento da mensagem pelo destinatário, essa forma de comunicação demanda um trabalho maior na concepção dos aplicativos, pelo fato de ser necessário criar outras formas de confirmar o recebimento das mensagens.

2.5.3 Domínio *point-to-point*

Domínio *point-to-point* (PTP) tem a característica de trabalhar com filas (*queues*) de mensagens (SUN, 2002). Nele, mensagens são criadas pela aplicação e enviadas para uma fila no gerenciador de mensagens, depois entregues, por esse, ao destinatário.

Uma questão a ser levada em consideração nesse domínio é o fato de que uma mensagem é entregue a uma fila específica (*queue*) e que essa fila só tem um destinatário, desta forma não é possível usar esse domínio quando se pretende compartilhar as mensagens com mais de um destinatário.

Portanto, o emprego desse domínio é interessante quando as mensagens a serem criadas têm um destinatário único, geralmente gerenciado pela mesma empresa. Por exemplo, uma empresa e sua filial que pretendem se comunicar por meio de mensagens. Dessa forma fica mais fácil ter certeza que apenas a filial está tendo acesso às mensagens enviadas.

2.5.4 Domínio *publish-subscribe*

No Domínio *publish-subscribe* (Pub/Sub) produtores de mensagens as publicam para um tópico específico, ao qual assinantes se conectam para retirar essas mensagens.

Trata-se de um sistema de compartilhamento abrangente, no qual uma mensagem pode ser entregue a um número indefinido de destinatários, sendo eficaz quando se pretende divulgar as mensagens sem saber exatamente quem as receberá.

Nesse domínio é mais conveniente ser usada comunicação assíncrona, pois a espera por respostas dependeria de todos os destinatários envolvidos, o que

aumentaria a espera à medida que novos destinatários se inscrevessem nos tópicos.

A configuração do domínio deve ser executada no servidor Glassfish, isto pode ser feito utilizando a interface web ou via código de deploy do Middleware produtor. Para que um cliente possa se inscrever e receber mensagens point-to-point ou publishe/subscribe, as configurações precisam ter sido realizadas anteriormente.

2.6 Trabalhos Relacionados

Sistemas de banco de dados distribuídos é um tema recorrente, sendo abordado em diferentes livros e artigos como em: Date (2004), Ray (2009), Ozsu e Valduriez (2011), dentre outros.

A consistência de dados em banco de dados em nuvem é um tema discretamente abordado na literatura cientifica (FREITAS, 2014). Mesmo assim é possível citar trabalhos que tratam do tema.

Os trabalhos expostos têm relevantes contribuições sobre o tema proposto por esta pesquisa, sendo analisados de acordo com sua proposta de solução para o problema da manutenção da consistência de dados em banco de dados distribuídos e foram selecionados por proporem suas soluções baseadas numa hierarquia entre as réplicas do banco de dados.

Em WANG (2010), o autor defende a divisão da estratégia da consistência em quatro categorias: da consistência forte à fraca além de combinações dessas de acordo com a frequência de leitura e escrita. Ajustando assim o nível de consistência dos dados em tempo de execução, de acordo com métricas de leitura e escrita previamente determinadas.

Não há participação do usuário na escolha do nível de consistência com o qual os dados devem ser tratados. Desse modo, o conhecimento do usuário sobre

sua aplicação é ignorado na escolha do nível de consistência, tendo importância apenas no momento de definir as métricas de leitura e escrita.

A vantagem dessa abordagem é o ajuste, em tempo de execução, dos níveis de consistência dos dados, tornando isto transparente ao usuário, por outro lado, o ponto, fraco dessa abordagem é a definição das métricas necessárias para esse ajuste, que fica a cargo de um especialista nos dados.

Em FREITAS (2014), a autora propõe o uso de réplicas do banco de dados, das quais uma réplica denominada *master* fica encarregada de receber atualizações e criar uma mensagem incorporando as atualizações que serão enviadas ao servidor *Glassfish.* Este servidor que se encarrega de entregá-la a seus destinatários (réplicas) de forma assíncrona. Os destinatários são providos de aplicação JAVA criada conforme especificação JMS (SUN, 2002).

Para garantir um baixo tempo de resposta, é proposto o uso de uma consistência parcial dos dados por meio da seleção dos dados que devem ser atualizado imediatamente e dos que podem esperar uma janela de tempo para sua atualização.

A proposta é interessante e adequada a muitas aplicações. No entanto, a parcialidade de sua consistência é um ponto negativo, tendo em vista que algumas aplicações importantes no mercado não poderiam utilizar esse tipo de garantia de consistência.

Para ajustar sua proposta a um grupo maior de aplicações, pretende-se descentralizar as atualizações da réplica *master*, garantindo sua aplicação a todas as réplicas, sem uma ordem definida de prioridade. Será retirada do SGBD Oracle[8] a responsabilidade de criar e enviar as mensagens ao *GlassFish*, sendo essa operação responsabilidade da aplicação. Assim, as réplicas, como clientes do sistema de mensagem, concorrem simultaneamente para aplicação das atualizações.

[8] http://docs.oracle.com/en/database/

Contudo esse método não garante que num determinado momento as réplicas estarão num estado consistente, sendo possível, por alguns instantes uma desatualização entre elas, levando em consideração que algumas réplicas estarão mais distantes, fisicamente, do servidor *Glassfish* que as outras, gerando retardos na realização de atualizações nestas. Sendo assim é importante que esse retardo seja o menor possível, para não causar problemas aos usuários.

Outra característica deste trabalho é que cada tabela está associada a um tópico específico. Com esta abordagem cada transação terá de criar uma mensagem para cada tabela envolvida na mesma. Por exemplo, caso uma transação envolva atualização em quatro tabelas, esta transação terá de criar e enviar quatro mensagens que serão administradas pelo servidor e entregues aos destinatários, isso geraria um fluxo maior de mensagens sendo trafegadas na rede.

Em seu trabalho Z*HAO* (2012) afirma que replicação de dados é uma estratégia para alcançar disponibilidade, escalabilidade e melhoria de desempenho no gerenciamento de dados. Segundo o autor existem duas arquiteturas, comumente usadas, de replicação de dados em nuvem, a arquitetura *master-slave* (mestre-escravo) e a *multi-master* (multi-mestre).

A arquitetura *multi-master* permite que cada réplica mantenha uma cópia completa do banco para servir a transações de leitura e escrita. Conflitos de escrita são resolvidos pelo *Middleware* de replicação. Por outro lado, a replicação *master-slave* melhora o rendimento de leitura. Nessa arquitetura, transações de leitura são servidas pelas réplicas *slaves* e transações de escrita pela *master*. O *Middleware* fica encarregado de repassar às réplicas *slaves* as transações de escrita executadas na réplica *master*, desta forma conflitos de gravação são resolvidos no lado *master*.

Seu estudo se concentrou na arquitetura *master-slave*, por ser a mais empregada por muitas aplicações web, apresentando um *framework* para a

replicação dos dados o qual deve considerar um *SLA* de atualidade dos dados para cada réplica.

O *SLA* (*Service Level Agreement* - Acordo de Nível de Serviço) trata-se de um contrato especificando regras a serem cumpridas por partes envolvidas num contrato. No contexto de sua pesquisa o *SLA* define o atraso aceitável de atualizações das réplicas do bando de dados.

O *framework* dispõe de três módulos: monitor, controle e ação. O primeiro monitora o atraso na atualização de cada réplica (diferença entre os *timestamp* de atualização no *master* e na réplica), o segundo, deve comparar os atrasos à SLA e disparar um comportamento do modulo de ação (terceiro).

O modelo trata apenas da atualização dos dados, não garantindo consistência, sendo esta indefinida se forte ou fraca. Ademais a definição do SLA pode ser bastante custosa, já que fica por conta do usuário definí-la.

A arquitetura de replicação *master/slave* adotada apresenta um risco considerável à disponibilidade pelo fato de que se a réplica *master* estiver indisponível não será possível realizar transações de escrita, uma vez que o *middleware* responsável não pode executar escrita diretamente nas réplicas *slaves*. Isso faz com que uma partição na rede no *host* da réplica *master* torne indisponível, pelo menos para escrita, todo o banco de dados.

A proposta deste trabalho é usar a arquitetura *multi-master*, com escrita realizada por meio de envio de mensagens, usando um sistema de mensageria criado em Java, o qual enviará uma mensagem contendo a escrita para um servidor *Glassfish*, que repassará a mensagem às réplicas que de fato executarão a operação de escrita em suas respectivas cópias do banco de dados.

Uma vez adotada a arquitetura *multi-master*, o banco de dados estará disponível para leitura e escrita mesmo que uma ou mais réplicas sofram uma partição de rede, contanto que ao menos uma réplica continue ativa.

O Quadro 2.2 faz uma comparação dos trabalhos relacionados, levando em consideração o método, o tipo de consistência e a participação do usuário na definição das métricas.

Quadro 2.2 – Comparação dos trabalhos relacionados

Autor	Método	Consistência	Interferência do Usuário	Vantagem	Desvantagem
Wang. Al 2010	Seleção de nível de consistência em tempo de execução. Quatro categorias são definidas: da consistência forte a fraca além de combinações delas, de acordo com a frequência de leitura e escrita.	**Indefinida**	Usuário não define a consistência , porém define as métricas para os ajustes em tempo de execução	Ajuste dos níveis de consistência em tempo de execução, tornando essa operação transparente para o usuário.	A Definição das métricas necessárias para os ajustes dos níveis requer um especialista no negócio.
Freitas et. Al 2014	Réplica os dados em réplicas hierarquicas, propagando atualizações da réplica *master* às *slaves* mesclando atualização ansiosa e preguiçosa de acordo com requisitos de negócios. Requer interferência de um especialista nos dados para definir quais dados devem ser mantidos consistentes em tempo de execução e quais podem estar em um estado inconsistente por um intervalo de tempo arbitrário.	**Fraca**	Usuário interfere na seleção dos dados que requerem consistência forte	Uso de sistema de mensageria para propagar as atualizações entre as réplicas do banco de dados.	A centralização das operações de atualizações em uma réplica master.
Zhao et. Al 2012	Uso de um *framework* que deve equilibrar a carga de usos das réplicas afim de garantir latência reduzida no consumo do SGBD. O mesmo requer participação de usuário para definir SLA que deve ser consultada para verificar se as atualizações estão dentro do esperado.	**Indefinida**	Usuário interfere definindo o SLA	Definição de uma SLA para avaliar se uma réplica deve ser desativadas caso ultrapasse um detrinado tempo para ser atualizada.	A centralização das operações de atualizações em uma réplica master.

Fonte: Elaborado pelo Autor (2017)

2.7 Conclusões do Capítulo

Este capítulo contribuiu para a dissertação apresentando uma revisão da literatura, fundamentando, assim, este trabalho. Foram abordados os temas de Banco de Dados distribuídos, Consistência de Dados considerando o teorema CAP, PACELC e tipos de consistência de dados, além desses temas foi abordado o paradigma de Middleware Orientado a Mensagens.

A revisão da literatura destacou a relevância do tema desta pesquisa e mostrou como os autores o abordaram. Mostrou ainda a necessidade de desenvolver novos métodos para prover a manutenção da consistência dos dados distribuídos, deixando de simplesmente ignorá-la.

O próximo capitulo apresenta o modelo proposto para manutenção de consistência de dados relacionais distribuídos, detalhando seus componentes, funcionalidades e arquitetura.

3 MODELO PARA MANUTENÇÃO DE CONSISTÊNCIA DE DADOS EM BANCOS DE DADOS RELACIONAIS DITRIBUÍDOS

Neste capitulo será apresentada a principal contribuição desta pesquisa, um modelo de manutenção de consistência de dados em sistemas de banco de dados relacionais distribuídos. De acordo com o dicionário Webster[9], um modelo pode ser, entre outras coisas:

- Um padrão para algo a ser feito;
- Uma descrição usada para ajudar a visualizar alguma coisa que não pode ser observada diretamente; ou
- Uma projeção teórica de um sistema possível ou imaginário.

Sobre essas definições, Dieste et al. (2002) afirmam que no desenvolvimento de *software*, os modelos são utilizados para descrever como será o sistema de *software* que se destina a resolver um problema. Este é o caso do modelo aqui apresentado.

Este modelo parte do pressuposto que algumas aplicações requerem consistência forte em todos os seus dados. Por exemplo, um sistema de transações bancárias, como os hospedados na web ou em pontos de acesso (caixa rápido) ou um sistema de verificação da situação veicular. Não é aceitável que ao realizar uma transação bancária ou uma consulta ao emplacamento de um veículo, o usuário acesse uma cópia do banco de dados desatualizada, o que poderia acarretar prejuízos aos envolvidos.

A estratégia para atender a essa demanda é garantir a manutenção da consistência dos dados por meio de um sistema que empregue alguma forma de atualização ansiosa dos dados. O modelo aqui proposto usará comunicação em grupo para aplicar as atualizações nas réplicas do banco de dados.

[9] https://www.merriam-webster.com/dictionary/model

Desta forma, o conhecimento do desenvolvedor não é necessário, pois, nesse caso, considera-se que todos os dados devem ser mantidos consistentes.

3.1 Modelo de Atualização Ansiosa Baseado em Comunicação em Grupo

Conforme discutido anteriormente (Seção 2.1.3), uma das estratégias utilizadas pelo modelo de atualização ansiosa tradicional é o *commit* em duas fases (2PC), a qual pode causar problemas de desempenho.

A partir desses estudos e mediante investigação de suas vantagens e desvantagens, o modelo aqui proposto foi desenvolvido, dispensando a utilização do *commit* em duas fases. No lugar do 2PC, o modelo proposto utiliza a comunicação em grupo, por meio de *Middleware Orientado a Mensagem* (MOM). Este sendo um canal de comunicação que transporta dados em um formato específico (mensagens), e permite a troca desses dados entre aplicações utilizando uma estrutura de dados própria para armazená-los.

A abordagem MOM escolhida para este trabalho foi a *publish-subscribe*. Nesse modelo, assinantes (*subscribers*) registram-se em tópicos de seu interesse e são notificados, de forma assíncrona, de mensagens enviadas a esse tópico pelos publicadores (*Publisher*). Isto é: as aplicações mantém comunicação por meio de mensagens que são armazenadas pelos *publishers* em tópicos e retiradas pelos interessados. Por sua vez um tópico é uma estrutura de dados semelhante a uma fila, diferenciando-se dessa por permitir que mais de um *subscriber* retire dela as mensagens.

Cada tópico, no modelo *publish-subscriber*, possui um assunto, isto é, cada tópico armazena mensagens com assuntos em comum. Cada *subscriber* assina os tópicos que tratem de assuntos de seu interesse (EUGSTER, 2007). No modelo aqui proposto, cada tópico deverá conter dados de uma transação. Por exemplo, o

tópico *jms/sigaProcesso* deverá conter dados de todas as mensagens destinadas à transação responsável por cadastrar um processo.

O uso do MOM permite que o sistema continue funcionando caso o *publisher* ou *subscriber* falhe, isto porque a troca de mensagens ocorre sem que as partes envolvidas se conheçam:

Para que a troca de mensagens aconteça é necessário, apenas, que as partes envolvidas (*publisher* e *subscriber*) conheçam o endereço onde o tópico está localizado. Informações como número de envolvidos podem mudar sem interferir no andamento da comunicação. Essa capacidade de aumentar ou diminuir o número de *subscribers* sem interferir na comunicação viabiliza a replicação dinâmica dos dados. Desta forma, a criação de novas réplicas do banco de dados pode ocorrer naturalmente, sem necessidade de interrupção do serviço.

Em uma visão geral, o modelo funciona da seguinte forma: existem duas camadas de aplicações desenvolvidas sob o paradigma de MOM, aqui denominadas *Middleware* N1 e *Middlware* N2. Cada transação executada no sistema cria um vetor (*ArrayList* na linguagem de programação Java) contendo os comando SQL a serem executados pelos SGBD das réplicas do banco de dados. Esse vetor é enviado para o *Middleware N1*, que, o recebendo, prepara-o para adicioná-lo a uma mensagem e depois envia essa mensagem para o *Provider* que armazena a mesma em um tópico determinado e notifica os clientes subscritos nesse tópico da chegada dessa mensagem.

Quando uma mensagem é publicada em um tópico, os interessados (*subscribers*), que aqui representam o *Middleware N2*, são notificados e realizam a retirada da mensagem do tópico, tratamento de seu conteúdo e aplicação dos comandos em suas réplicas.

O modelo foi desenvolvido e testado com uso da aplicação *sigaProcesso* da Universidade Federal de Pernambuco (UFPE. O referido sistema utiliza um Servidor Web *Tomcat* e, por conta disso, não pode se comunicar diretamente ao Servidor Web *Glassfish*, utilizado aqui como *Provider* do MOM. Para resolver esse

problema foi desenvolvido um *Web Service* destinado a prover um meio de comunicação entre o *sigaProcesso* e o *Middleware N1*. Isso torna-o aplicável para um grupo de aplicações que utilizam um servidor web diferente do Glassfish, que se configura numa vantagem. Caso uma aplicação utilize o Glassfish como servidor web, basta retirar o web service e utilizar seu Glassfish fazendo a comunicação direta.

O modelo proposto neste trabalho tem o objetivo de lidar com partições que impeçam que uma réplica do banco de dados seja atualizada, ou seja, uma mensagem foi envida até o *Provider* e esta deve ser entregue aos middleware responsáveis por aplicá-la às réplicas, mesmo que estes middleware não estejam aptos no momento. Outras partições como erro no glassfish não foram contempladas neste trabalho. No entanto redundância do servidor glassfish poderia resolver esse problema, artifício que não foi testado neste trabalho.

Os detalhes da arquitetura dos *middleware* serão apresentados na seção seguinte.

3.2 Arquitetura do Modelo: Uma Visão Geral

Esta seção tem o objetivo de descrever o modelo proposto, detalhando sua arquitetura e funcionamento.

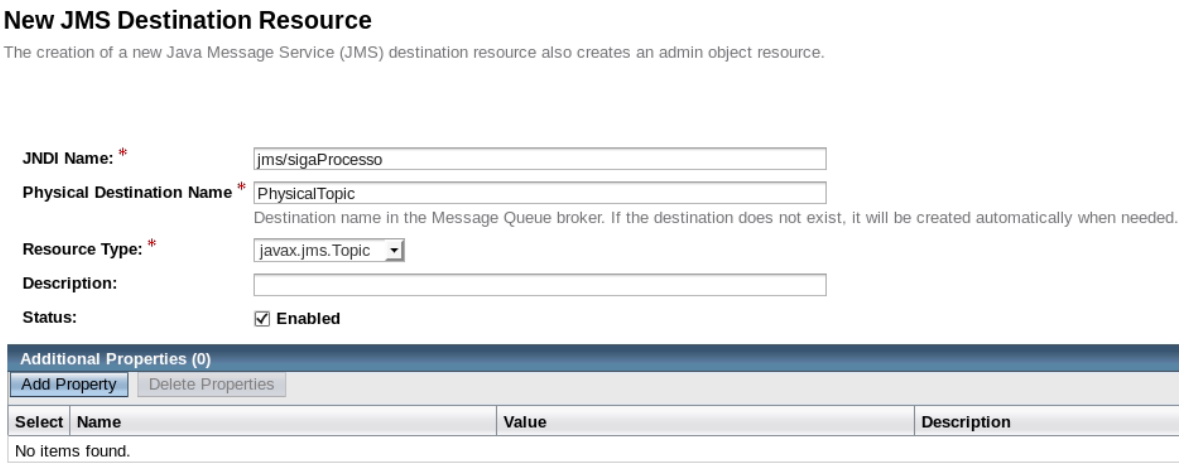
3.2.1 *Middleware N1*

O *Middleware N1* tem as seguintes funções: (1) Criar os *Tópicos*; (2) Receber as transações vindas da aplicação; (3) Encapsular seus dados em um formato de mensagem aceito pelo *MOM*; e (4) Enviar as mensagens ao *Provider*

para que o mesmo armazene-as nos tópicos adequados e notifique os interessados.

A criação dos *Tópicos* acontece no momento que a aplicação é implantada (*deploy*) no servidor web. Uma vez criados os Tópicos, não será necessário mais recriá-los, pois os mesmos ficam registrados no servidor web. Apesar de ser uma das funções do N1, a criação dos tópicos poderia ser realizada pelo usuário, diretamente no servidor web, através de sua *GUI* (Graphical user interface – Interface Gráfica do Usuário em português), conforme Figura 3.1.

Figura 3.1 Interface gráfica de criação do tópico jms/sigaProcesso



Fonte: Elaborada pelo Autor (2017).

Para que o *Middleware N1* possa enviar mensagens a um tópico é necessário também criar um *ConnectionFactory,* objeto que encapsula um conjunto de parâmetros de configuração de conexão usado pelos clientes para criar conexões com os provedores JMS (SUN 2002). Esses objetos dão suporte a criações simultâneas de conexões, o que possibilita que mais de um cliente possa utilizá-lo simultaneamente.

O *ConnectionFactory*, neste modelo, é criado via aplicação no momento da implantação da mesma no servidor web Glassfish.

A segunda função do *N1* é receber as transações vindas da aplicação. Para executar esta função é utilizado um Web Service, conforme explicado na Seção 3.1.

A terceira função do N1 é encapsular os dados recebidos em um formato de mensagem aceito pelo MOM. Para esta pesquisa foi utilizada uma estrutura de dados nativa da linguagem de programação Java denominada *ArrayList.* Esta trata-se de um objeto que guarda valores como um vetor de chaves/valores, no qual cada índice do vetor corresponde à localização de um valor específico. Esse vetor, criado pela aplicação e enviado ao N1, contém em seus valores uma cadeia de caracteres (*String*) que corresponde aos comandos de atualização do banco de dados.

Este formato (ArrayList de String) não é reconhecido como um formato válido pelo MOM. Para enviar esses dados para o *Provider* é necessário transformá-lo em objeto do tipo *ObjectMessage*, o qual pode encapsular um objeto Java em seu conteúdo. Esse encapsulamento é suficiente para que os dados recebidos da aplicação possam ser enviados, como conteúdo de uma mensagem, para o *Provider*.

Uma vez encapsulados os dados num formato aceito pelo MOM, o *N1* envia as mensagens para o *Provider*, sua quarta função.

O *Provider* faz o armazenamento das mensagens e a notificação dos clientes, sendo transparente para o desenvolvedor como isto acontece. O *Provider* utilizado nesta pesquisa é executado no servidor web *Glassfish*.

Outro detalhe a ser observado é o modo como as mensagens chegarão aos destinatários. Há duas formas utilizadas pelo *MOM* para notificar os clientes de que há mensagens no tópico: ***Pull*** e ***Push***.

No primeiro, o cliente fica responsável por verificar, em intervalos determinados pelo usuário, se há alguma mensagem armazenada no tópico de interesse. No segundo, o *Provider* se encarrega de notificar aos clientes da chegada de novas mensagens. A forma de notificação *Push* foi selecionado por

automatizar a notificação do cliente utilizando um ouvinte (*listener*) implementado em cada cliente que fica aguardando por notificações do *Provider*. Essa forma garante que uma mensagem não ficará no tópico indefinidamente esperando que um cliente verifique sua chegada.

3.2.2 *Middleware N2*

O *Middleware N2* tem as seguintes funções: (1) Subscrever-se no tópico de interesse; (2) Aguardar notificações do *Provider* da chegada de novas mensagens no tópico de interesse; (3) Retirar as mensagens do tópico; (4) Aplicar as transações (contidas na mensagem) na réplica do banco de dados.

Para realizar a primeira função, *N2* deve enviar uma mensagem ao *Provider* contendo o nome do tópico de interesse. Esse procedimento acontece quando uma aplicação cliente é implantada (*deploy*) no servidor web *Glassfish.* Para que tudo ocorra sem erros, o *Tópico* deve ter sido criado no provider pelo *Middleware* N1 quando este implantado ou via interface gráfica (Seção 3.2.1).

A segunda função é desempenhada por um processo ouvinte (*Listener*) que aguarda notificações do *Provider*. Esse ouvinte implementa o método ***Push*** explicado na Seção 3.2.1. Quando uma mensagem é armazenada, o *Provider* se encarrega de notificar a esse processo que uma nova mensagem foi armazenada no Tópico.

Sua terceira função é retirar as mensagens do tópico, o que acontece após receber uma notificação. Essa retirada segue a ordem do armazenamento, na qual a primeira mensagem a ser armazenada é a primeira a ser retirada.

A respeito dessa retirada, a classe *ReceiveMessage*, responsável por receber mensagens nos cliente, implementa a classe *MessageListener*, classe abstrata nativa da especificação JMS que fornece uma assinatura de um método (onMessage) responsável por receber as notificações do *Provider* e retirar as mensagens do tópico.

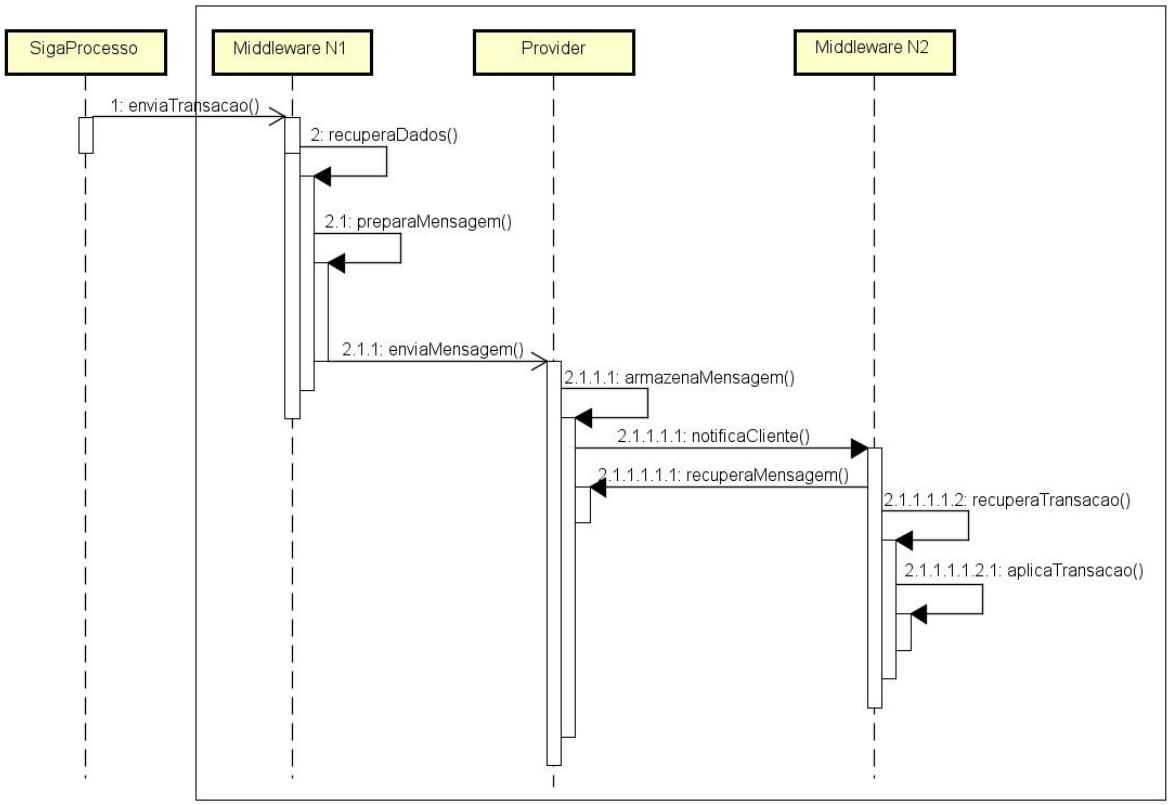
A quarta função é a aplicação das atualizações (inserção, remoção ou alteração) na réplica do banco de dados. A forma serial de aplicação garante a ordem das atualizações nas réplicas do banco de dados.

3.2.3 Interações entre os *Middleware N1 e N2*

Está Seção se propõe a detalhar as interações entre os *Middleware N1* e *N2* para execução de suas funções.

O Diagrama de Sequência (Figura 3.2) mostra, de forma simples, as interações entre os *Middleware*, estas interações representam o melhor caso.

Figura 3.2 – Diagrama de Sequência das Interações entre os *Middleware N1* e *N2*



Fonte: Elaborada pelo Autor (2017)

Como pode ser visto na Figura 3.2, a aplicação inicia o processo com uma chamada ao *Middleware N1*. Após essa chamada o *N1* procede com o tratamento do conteúdo recebido com a chamada. Esse conteúdo deve ser organizado em um formato de mensagem aceito pelo MOM. Criada a mensagem, *N1* a envia ao *Provider*, finalizando assim sua participação no processo. Observa-se que a partir deste momento, o *N1* deixa de ter responsabilidades quanto a manutenção da mensagem e sua entrega aos destinatários. Isso ficará por conta do *Provider*, o qual armazenará as mensagens, notificará os interessados e gerenciará a exclusão da mensagem após todos interessados subscritos a tenham retirado do tópico.

Após ser notificado, *N2* retira do tópico a mensagem e realiza os procedimentos descritos na Seção 3.2.2.

As interações entre estes Middleware acontecem por meio do *Provider*, não havendo nenhuma comunicação direta entre eles, garantindo assim baixo acoplamento, o que é uma vantagem desse paradigma (MOM), ou seja, não existe dependência de funcionamento entre os *Middleware*, cada um executa suas funções sem necessitar que o outro esteja disponível. Ou seja, *N1* pode publicar suas mensagens sem que *N2* esteja apto a recuperá-las e *N2* pode recuperar mensagens, mesmo que *N1* tenha sofrido uma partição do serviço.

3.3 Conclusões do Capítulo

Este capítulo apresentou o modelo que visa garantir a manutenção da consistência de dados armazenados no ambiente de banco de dados distribuído.

Foram explanados os detalhes do funcionamento do modelo destacando, também, alguns detalhes das tecnologias empregadas.

Adicionalmente, foram expostos os aspectos arquiteturais relacionados ao modelo, inclusive descrição dos *Middleware* N1 e N2.

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 dos *Middleware N1* e *N2*.

4 PROTÓTIPO PARA APLICAÇÃO DO MODELO

Este capítulo tem o objetivo de apresentar o protótipo desenvolvido na linguagem de programação Java[10] para implementar o modelo proposto nesta pesquisa. São destacados o ambiente e as tecnologias empregadas para seu desenvolvimento.

4.1 Minimundo

Antes de apresentar as etapas de desenvolvimento do sistema, convém expor o minimundo criado para testar o protótipo e que será também utilizado como ilustração, exemplificando as atividades práticas a serem realizadas.

O minimundo é a aplicação *sigaProcesso*, parte integrante do sistema de gestão acadêmica *SIG@*[11] utilizado na UFPE desde 2003. Para os testes a serem realizados serão consideradas quatro (4) tabelas, como pode ser visto na Figura 4.1, tendo a tabela *siga_processo* como principal.

Criada e mantida na linguagem de programação Java essa aplicação (*sigaProcesso*) é suficiente para os testes necessários para comprovação da eficácia do modelo proposto nesta pesquisa, por ser possível alterações na mesma, afim de ajustá-la ao paradigma de comunicação por meio de mensagens, sem muitas dificuldades.

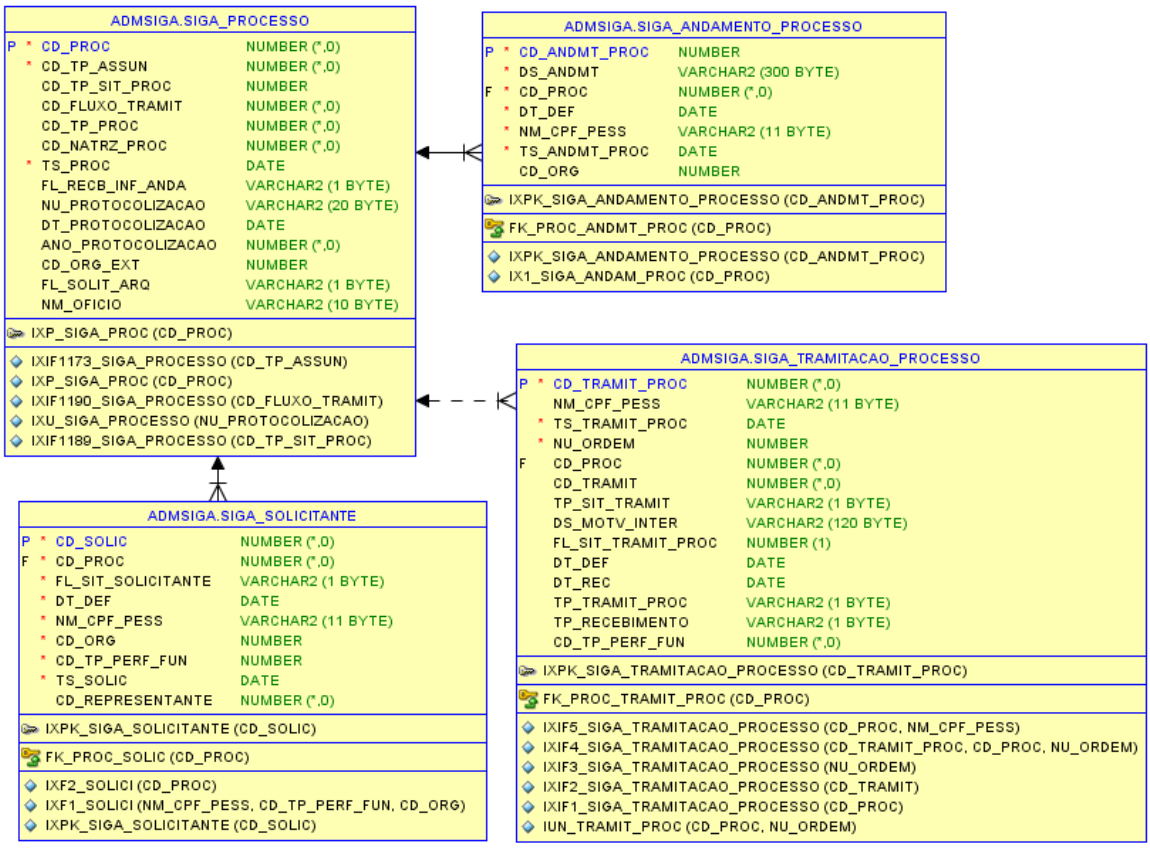
As tabelas *siga_processo*, *siga_andamento_processo*, *siga_tramitacao_processo* e *siga_solicitante* (Figura 4.1) são suficientes para representar um processo e para esta pesquisa se apresentam necessárias para a realização dos testes necessários.

---

[10] http://www.oracle.com/technetwork/java/javaee/documentation/index.html

[11] https://siga.ufpe.br/ufpe/jsp/acesso/pages/inicio.jsf

Figura 4.1 – Diagrama Entidade Relacionamento proposto



Fonte: Elaborada pelo Autor (2017).

As seções seguintes têm o objetivo de apresentar o desenvolvimento do protótipo que implementa o modelo proposto para a realização das tarefas descritas no capitulo anterior.

4.2 Configuração das Réplicas do Banco de Dados

O sistema gerenciador de banco de dados utilizado pelo *sigaProcesso* é o Oracle[12] 11.2.0 (Oracle 11G), na versão *standard*. Para esta pesquisa foi utilizada uma cópia do banco de dados do *SIG@* replicado em três réplicas com o uso do aplicativo de *backup* e *recovery*, nativo do SGBD Oracle, *RMAN*[13], aplicativo utilizado para criação e recuperação de backup pelo SGBD Oracle.

Apesar de o protótipo ser desenvolvido para operar atualizações nesse SGBD, não está limitado a este, pois o mesmo (SGBD) se comporta passivamente no processo de aplicação das atualizações, sendo assim qualquer SGBD poderia ser utilizado, tendo apenas que fazer os devidos ajustes quanto à conexão entre o protótipo e o banco de dados.

Com a ferramenta *RMAN* foi gerado um backup do banco de dados de produção do *SIG@*. Esse backup foi utilizado para a criação de cópias do banco. Como as réplicas foram criadas utilizando a mesma cópia de backup, continham o mesmo estado de dados.

As réplicas estão hospedadas em três hosts dedicados exclusivamente para tanto, dois localizado no *datacenter* do NTI-UFPE[14] em máquinas virtuais criadas com o VmWare[15], o terceiro host localizado no datacenter do CIn-UFPE.

---

[12] http://docs.oracle.com/en/database/ Acesso: 29/07/2016

[13] https://docs.oracle.com/cd/B19306_01/backup.102/b14191.pdf

[14] https://www.ufpe.br/nti/index.php?option=com_content&view=article&id=88&Itemid=71 Acesso: 02/08/2016

[15]https://www.vmware.com/support/pubs/vsphere-esxi-vcenter-server-pubs.html Acesso: 02/08/2016

4.3 Ajustes no sigaProcesso

Como o sigaProcesso, sendo integrado ao *SIG@*, é gerenciado por um servidor web ***Tomcat***[16], não podendo, por isso, se comunicar diretamente com o *Middleware N1* o qual por sua vez é gerenciado por um servidor web *Glassfish*, foi desenvolvido e disponibilizado um Web Service no *host* que hospeda o *Glassfish* para possibilitar a comunicação com o sigaProcesso. Esse Web Sevice será detalhado na Seção 4.4.1.

Além do consumo desse *Web Service*, o fluxo da operação do *sigaProcesso* também sofreu mudanças que consistem em não persistir os dados através da conexão usada pelo aplicativo, mas sim armazenar os mesmos em uma estrutura de dados nativa da linguagem Java chamada ArrayList, um vetor de valores que pode armazenar o tipo de dados especificado na sua criação, nesse caso o tipo de dados é *String*, que representa uma cadeia de caracteres.

No cadastro de um processo, como o requerimento de auxílio natalidade por exemplo, quatro (4) tabelas são envolvidas: siga_processo, siga_tramitacao_processo, siga_andamento_processo e siga_solicitante. Na forma natural as atualizações são operadas em cada uma das tabelas mencionadas, na ordem que estão dispostas acima. Ao finalizar e com todas alterações efetuadas com sucesso, um comando de *commit* é executado, confirmando definitivamente as alterações no banco de dados. Caso aconteça algum problema na atualização de alguma das tabelas, uma exceção é lançada e um comando de *rollback* é executado cancelando qualquer alteração realizada nas tabelas anteriormente à exceção, garantindo assim que nenhuma alteração seja confirmada definitivamente.

Ao invés de realizar as operações de atualizações diretamente no banco de dados, como é normal numa aplicação, os comandos são armazenados numa

[16] http://tomcat.apache.org/tomcat-7.0-doc/ Acesso: 29/07/2016

lista, esta lista é enviada para o middleware, através do web service, para que a mesma seja integrada a uma mensagem. Assim o sigaProcesso não executa, de forma direta, atualizações nas réplicas do banco de dados, mas passa o conteúdos das atualizações ao middleware responsável. Conclui-se a participação do sigaProcesso deste modo.

4.4 Configuração do Middleware *N1*

Como visto na Seção 3.2.1, o *Middleware N1* tem as funções criar tópicos, receber transações vindas da aplicação, organizar os dados recebidos em um formato de mensagem aceito pelo MOM, e enviar as mensagens ao *Provider.* Para atingir tais funções foi criada uma aplicação chamada ***MOMFirstLayer*** composta por um *Web Service* e um *Middleware Orientado à Mensagem*.

4.4.1 Web Service

Segundo Crasso (2011), *Web Service* são serviços projetados para dar suporte à interação entre fornecedor e consumidor por meio da Internet. É uma funcionalidade criada e disponibilizada na Internet por um fornecedor. O fato de usar a Internet, como meio de acesso, torna esses serviços muito úteis e utilizados na *Computação Orientada a Serviço,* emergente paradigma computacional.

Para esta pesquisa, o modelo WSDL (Web Service Description Language) foi utilizado para criar e disponibilizar serviços acessíveis remotamente. Trata-se de um formato padrão para descrever serviços hospedados na web e prestados de acordo com um contrato..

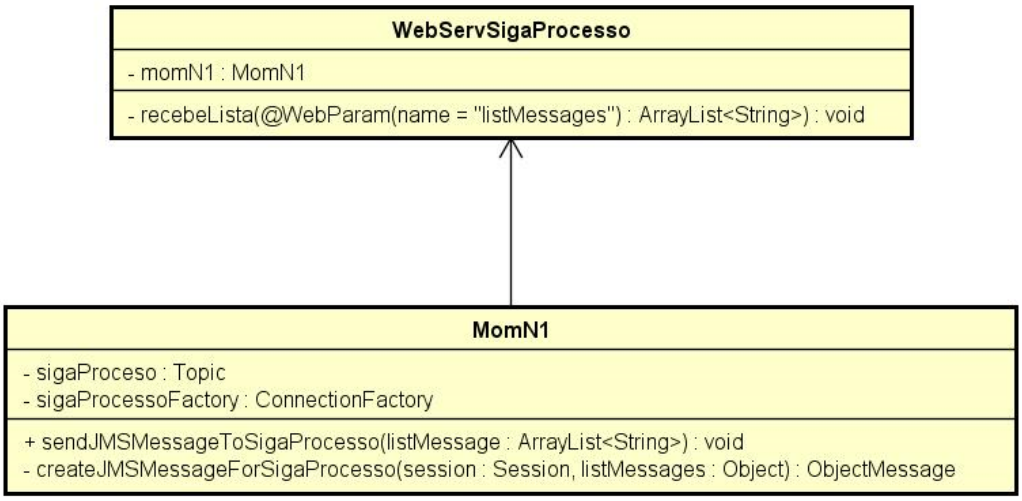
Para esta pesquisa um contrato entre o *sigaProcesso* e o *Middleware N1* foi estabelecido conforme modelo WSDL. Desta forma, o método *recebeLista* da classe *WebServSigaProcesso* pode ser acessado, via Internet, pelo *sigaProcesso*,

como se fizesse parte do mesmo. Desta maneira a lista de comandos SQL é passada para o *N1*.

4.4.2 Aplicação *MOMFirstLayer*

Responsável pela criação das mensagens, foi desenvolvido como um protótipo na linguagem de programação Java, usando a *api* JMS criada pela Sun. Basicamente recebe uma requisição do *sigaProcesso* por meio do *Web Service*, trata os dados criando uma mensagem e envia a mesma ao *Provider* para publicação no Tópico. A Figura 4.2 apresenta o diagrama de classes do MOMFirstLayer.

Figura 4.2 – Diagrama de Classes do MOMFirstLayer



Fonte: Elaborada pelo Autor (2017).

O método *recebeLista* da classe *WebServSigaProcesso* é responsável por receber chamados da aplicação *sigaProcesso*, por meio do Web Service, e encaminhar o conteúdo do chamado à classe MomN1, a qual desempenha as funções deste *Middleware.* É nesta classe que o *Middleware* responsável por

tratar os dados, criar mensagens e enviar para o *Provider* foi desenvolvido. A Figura 4.3 detalha a implementação desta classe.

Figura 4.3 – Representação visual da classe MomN1

```
17  public class MomN1 {
18      @Resource(mappedName = "jms/sigaProcesso")
19      private Topic sigaProcesso;
20      @Resource(mappedName = "jms/sigaProcessoFactory")
21      private ConnectionFactory sigaProcessoFactory;
22      public void sendJMSMessageToSigaProcesso(ArrayList<String> listMessages) throws JMSException {
23          Connection connection = null;
24          Session session = null;
25          try {
26              connection = sigaProcessoFactory.createConnection();
27              session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
28              MessageProducer messageProducer = session.createProducer(sigaProcesso);
29              messageProducer.send(createJMSMessageForjmsSigaProcesso(session, listMessages));
30
31          } finally {
32              if (session != null) {
33                  try {
34                      session.close();
35                  } catch (JMSException e) {
36                      Logger.getLogger(this.getClass().getName()).log(Level.WARNING, "Cannot close session", e);
37                  }
38              }
39              if (connection != null) {
40                  connection.close();
41              }
42          }
43      }
44      private ObjectMessage createJMSMessageForjmsSigaProcesso(Session session, Object listMessages) throws JMSException {
45          ObjectMessage list = session.createObjectMessage();
46          list.setObject((Serializable) listMessages);
47          return list;
48      }
49  }
```

Fonte: Elaborada pelo Autor (2017).

Conforme pode ser observado na Figura 4.3, o Middleware *N1* recebe, do Web Service, um objeto do tipo ArrayList de String, por meio do método ***sendJMSMessageToSigaProcesso*** (linha 22). Para manter comunicação com o *Provider* cria os objetos *connection* e *session*, para a criação da mensagem cria também um objeto do tipo *MessageProducer* que será utilizado para produzir uma mensagem. O método ***createJMSMessageForjmsSigaProcesso***, executado na linha 29 como parâmetro do método ***send***, este nativo da especificação JMS, cria um objeto do tipo ***ObjectMesssage*** e encapsula nele, serializada, a lista de comandos.

O método ***send*** envia a mensagem para o tópico *sigaProcesso*, definido na criação da mensagem (linha 28) de forma transparente, sem participação do programador.

4.5 Configuração do Middleware N2

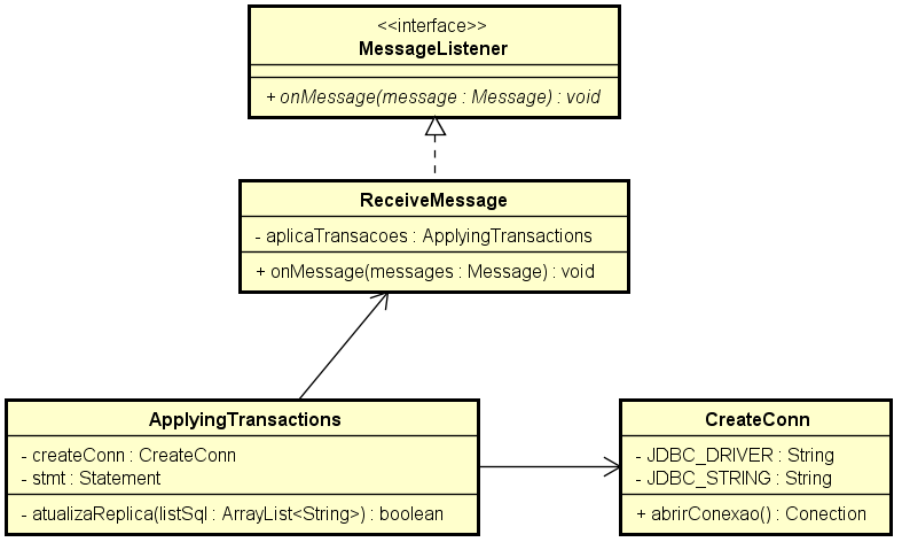
Como foi visto na Seção 3.3.2, o *Middleware N2* tem as funções de subscrever-se no Tópico, aguardar notificações do *Provider*, resgatar mensagens publicadas no tópico de interesse, decodificá-las e executar os comandos extraídos em sua réplica do banco de dados.

Uma aplicação chamada *MOMSecLayer*, acrônimo para Message Oriented Middleware of Second Layer (Middleware Orientado a Mensagem da Segunda Camada, em português) foi criada para realizar essas funções.

4.5.1 Middleware *MOMSecLayer*

Responsável por recolher as mensagens publicadas no tópico *sigaProcesso* foi desenvolvido um protótipo na linguagem de programação Java, usando a *api* JMS conforme Diagrama de Classes da Figura 4.4.

Figura 4.4 – Diagrama de Classes do MOMSecLayer



Fonte: Elaborada pelo Autor (2017).

Como pode ser visto na Figura 4.4, a classe *ReceiveMessage* implementa a interface *MessageListener* (nativa do pacote JMS), a qual atua como um processador de eventos assíncronos para mensagens. *ReceiveMessage* implementa o método *onMessage* (método abstrato da classe *MessageListener*), o qual por sua vez recebe notificações, retira mensagens do tópico e executa ações para atualização das réplicas.

Como mostra a Figura 4.5, após receber uma mensagem e transformar seu conteúdo num ArrayList de *String* (linha 22), acessa o método *atualizaReplica*, do objeto *aplicaTransacao*, o qual realiza o procedimento para atualizar as réplicas do banco de dados.

Figura 4.5 – Classe ReceiveMessage, reponsável por recuperar mensagens publicadas no tópico sigaProcesso

```
import ...;

@MessageDriven(mappedName = "jms/sigaProcesso", activationConfig = {
    @ActivationConfigProperty(propertyName = "clientId", propertyValue = "MOMSecLayerCin"),
    @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "Durable"),
    @ActivationConfigProperty(propertyName = "subscriptionName", propertyValue = "jms/sigaProcesso"),
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic")
})
public class ReceiveMessage implements MessageListener {

    private final ApplyingTransactions aplicaTransacoes = ApplyingTransactions.getInstancia();

    public ReceiveMessage() {
    }

    @Override
    public void onMessage(Message messages) {
        try {
            ObjectMessage objMessages = (ObjectMessage) messages;
            ArrayList<String> listSql;
            listSql = (ArrayList<String>) objMessages.getObject();
            try {
                aplicaTransacoes.atualizaReplica(listSql);
            } catch (SQLException e) {
                try {
                    throw e;
                } catch (SQLException ex) {
                    Logger.getLogger(ReceiveMessage.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        } catch (JMSException | ClassNotFoundException | IOException jex) {
            System.out.println("Exception");
        }
    }
}
```

Fonte: Elaborada pelo Autor (2017).

4.6 Conclusões do Capítulo

Este capítulo apresentou o modelo proposto do ponto de vista técnico. Foram detalhadas a tecnologias empregadas, bem como algumas partes dos código, relevantes para o entendimento do funcionamento do modelo, as quais foram expostas em imagens e o seu funcionamento foi explicado.

O capítulo seguinte deste trabalho apresenta os experimentos realizados para mostrar o uso e a eficácia do modelo em manter a consistência de dados em banco de dados relacionais distribuídos.

5 EXPERIMENTOS

Este capítulo apresenta os testes executados por meio dos protótipos dos *Middleware* N1 e N2, os quais foram realizados em um ambiente criado com o objetivo de verificar sua efetividade na manutenção da consistência dos dados entre as réplicas.

5.1 Ambiente de Testes

Para realização foi criado um ambiente que visa simular uma situação real de cadastro de processos. Para tal foram preparadas cinco (5) máquinas virtuais distribuídas conforme mostra o Quadro 5.1.

Quadro 5.1 – Relação de Máquinas Virtuais utilizadas para validar o protótipo e suas localizações

Máquina	Função	Local
Host1	Hospedar a aplicação sigaProcesso	NTI/UFPE
Host2	Hospedar o Provider (Glassfish 4.1) e Middleware (MOM)	NTI/UFPE
Host3	Réplica 1 do banco de dados	NTI/UFPE
Host4	Réplica 2 do banco de dados	NTI/UFPE
Host5	Réplica 3 do banco de dados	CIn/UFPE

Fonte: Elaborado pelo Autor (2017).

Abaixo segue descrição das máquinas virtuais.

Host1: Criada com VirtualBox 5.1.8[17], Sistema Operacional Fedora 21[18], processador com dois (2) núcleos com 3.4GHz de processamento, Memória RAM com quatro (4) GB e disco rígido (HD) com 25 GB.

[17] https://www.virtualbox.org/wiki/Changelog

Host2: Criada com VirtualBox 5.1.8, Sistema Operacional Centos 6.8[19], processador com um (1) núcleo com 3.4GHz de processamento, Memória RAM com três (3) GB e disco rígido (HD) com 25 GB.

Host3: Criada com VmWare 6[20], Sistema Operacional Centos 6.8, processador com um (1) núcleo com 3.4GHz de processamento, Memória RAM com quatro (4) GB e disco rígido (HD) com 125 GB**.**

Host4: Criada com VmWare 6, Sistema Operacional Centos 6.8, processador com um (1) núcleo com 3.4GHz de processamento, Memória RAM com quatro (4) GB e disco rígido (HD) com 125 GB

Host5: Criada com XenServer 6.5[21], Sistema Operacional Centos 6.8, 4 GB de memória Ram e 120 GB de disco.

A decisão por utilizar cinco (5) máquinas virtuais se deu pelo fato de que não seria possível utilizar máquinas físicas e porque não estava em estudo o desempenho de máquinas virtuais comparados a máquinas físicas. Ainda sobre as máquinas, utilizar cinco (5) máquinas se mostrou suficiente para distribuir o banco de dados em três (3) réplicas e manter duas máquinas para hospedar o sigaProcesso e os middlewate N1 e N2.

Na Figura 5.1 tem-se um diagrama de rede que representa a infraestrutura formada pelas máquinas virtuais descritas.

---

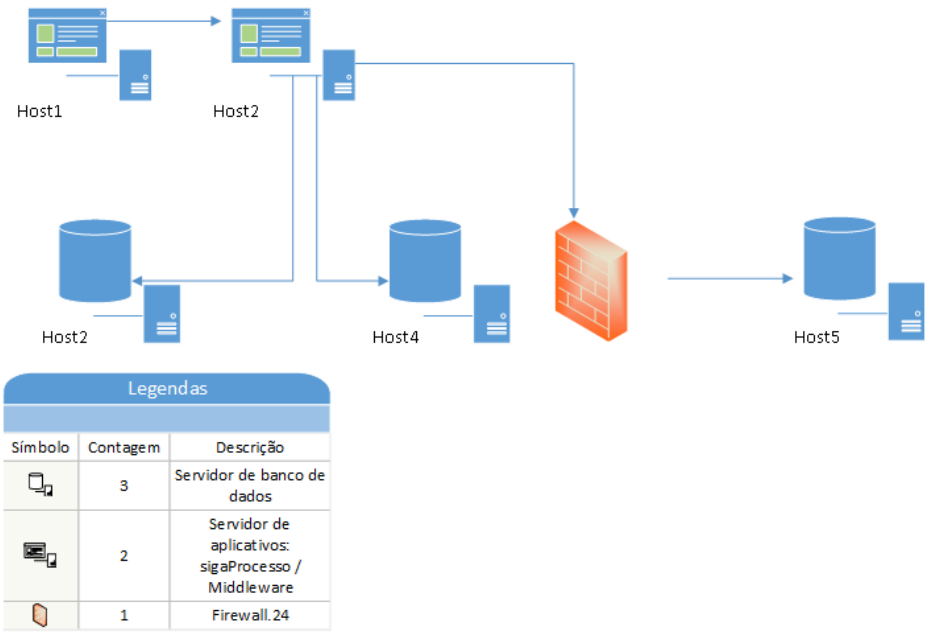
[18] https://docs.fedoraproject.org/en-US/Fedora/21/html/Release_Notes/index.html

[19] https://wiki.centos.org/Manuals/ReleaseNotes/CentOS6.8

[20] https://pubs.vmware.com/vsphere-60/index.jsp

[21] https://docs.citrix.com/en-us/xencenter/6-5.html

Figura 5.1 – Diagrama de Rede.



Fonte: Elaborada pelo Autor (2017).

5.1.1 Réplicas do Banco de dados

O sistema gerenciador de banco de dados (SGBD) utilizado foi o Oracle Database 11g Release 2 (11.2.0). Conforme mencionado anteriormente, este SGBD foi selecionado pelo fato de que o *sigaProcesso*, em estudo nesta pesquisa, utiliza o mesmo.

Porém, outro SGBD poderia ser utilizado, pois o protótipo criado está totalmente desacoplado do banco de dados, utilizando-o apenas como um repositório de dados.

Para a replicação do banco de dados foi utilizada a ferramenta *RMAN*[22] para criar um *backup* do servidor de produção e restaurá-lo nas réplicas, garantindo,

[22] http://docs.oracle.com/database/122/RCMRF/RMAN.htm#RCMRF111

assim, que as réplicas apresentariam um mesmo estado quando fossem disponibilizadas para o protótipo.

5.1.2 Sistema de Mensagens

O *Provider*, sistema de mensagens desenvolvido neste trabalho, tem a função de armazenar mensagens, gerenciar os tópicos e notificar os clientes (subscriberd). Ele está hospedado na mesma máquina virtual que os *Middleware N1 e N2*, apesar de não ser uma condição, pois o *Provider* poderia estar hospedado em outra máquina sendo acessado pela rede, A estratégia de hospedá-los na mesma máquina é interessante por melhorar o desempenho da comunicação entre os *Middleware* e o *Provider*, além disso gera apenas um ponto de risco de partição, o que seria diferente utilizando duas máquinas.

Foi desenvolvido utilizando o Java Message Service (JMS) na versão 1.1 e disponibilizado no servidor Glassfish 4.1. Sua principal vantagem está na confiabilidade do serviço prestado pela API JMS, a qual garante a entrega da mensagem apenas uma vez para cada cliente.

O servidor Glassfish ainda garante que caso falhe, sendo impossibilitado de entregar alguma mensagem, esta não será perdida, esta mensagem será armazenada e será entregue quando o serviço for restabelecido.

5.1.3 Ambiente Simulado

O ambiente de testes foi disponibilizado por um dia para sete (7) servidores da UFPE lotados no Núcleo de Tecnologia da Informação (NTI) para que os mesmos realizassem cadastros de processos. Durante esse dia foram cadastrados 130 processos. Esta parte dos testes serviu para verificar cadastros concorrentes, de modo a averiguar se o protótipo estava apto para lidar com

requisições simultâneas. Para confirmar que a ordem de entrega das mensagens foi respeitada, durante os cadastros, os *middleware* responsáveis por aplicar as atualizações nas réplicas foram, propositalmente, desativados por quinze minutos, a fim de verificar se as mensagens seriam entregues posteriormente e sendo entregues se seriam entregues na mesma ordem que foram criadas

Após a realização dos cadastros acima citados seguiram-se os testes para verificar operações de atualizações e exclusões. Estes testes foram realizados numa parte dos processos cadastrados com um total de vinte (20) registros. Ao fim de cada etapa, inserção, atualização e exclusão, consultas específicas foram realizadas para verificar a manutenção da consistência dos dados replicados.

Como o banco de dados considerado foi uma réplica do utilizado em produção, as tabelas envolvidas continham dados no início dos testes, sendo esses dados desconsiderados nas avaliações seguintes. O Quadro 5.2 mostra o número de registros contidos em cada tabela no início dos testes.

Quadro 5.2 - Quantidade de registros originais das réplicas

Tabela	Qtd
siga_processo	**873821**
siga_andamento_processo	**14568866**
siga_tramitacao_processo	**9190240**
siga_solicitante	**112898**

Fonte: Elaborado pelo Autor (2017).

Para novos registros foram utilizadas ***sequences*** (funcionalidade nativa do SGBD Oracle que gera números sequenciais) para criar a chave primária em cada tabela.

5.2 Avaliações

Para comprovação da efetividade do protótipo, foram realizadas consultas nas réplicas do banco de dados a fim de verificar o quanto essas réplicas se mantiveram consistentes. Primeiro foram realizadas consultas após a realização dos cadastros, depois realizadas após alterações executadas em alguns registros (20) e por último realizadas consultas após a exclusão de outros registros (10).

5.2.1 Consultas

A seguir serão apresentadas as consultas realizadas nas réplicas do banco de dados.

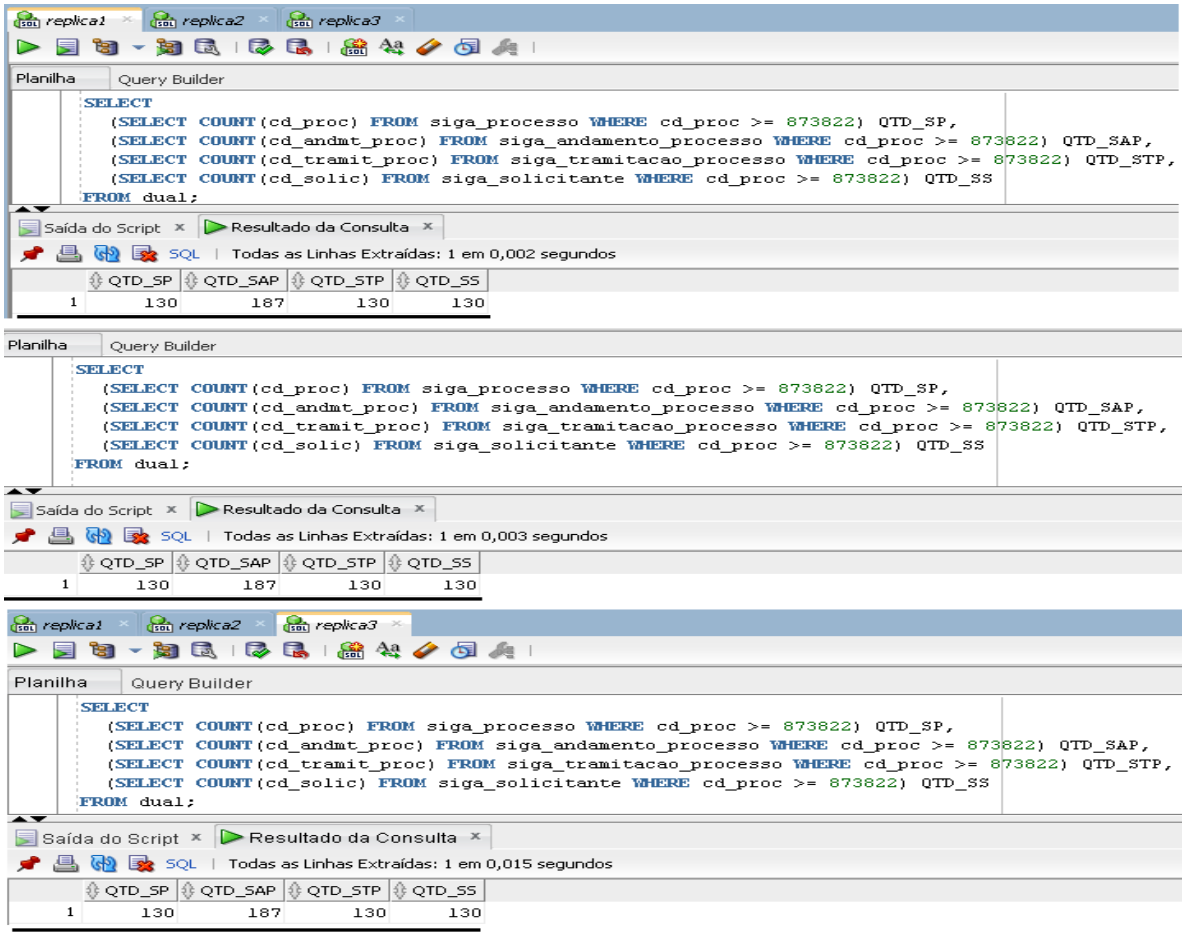
1ª Etapa

Após a realização dos cadastros seguiram-se os testes com consultas a fim de verificar se a consistência foi mantida entre as réplicas. Descritas a seguir.

Número de Tuplas

Levando em consideração que as réplicas devem, no final do experimento, ter o mesmo número de tuplas como requisito primário para a manutenção da consistência, essa consulta visa contar as tuplas nas tabelas de cada réplica.

Figura 5.2 – Consulta para contabilizar registros nas tabelas.



Fonte: Elaborada pelo Autor (2017).

Como pode ser observado, as três réplicas apresentam o mesmo número de registros nas tabelas envolvidas. Uma observação deve ser destacada para a tabela *siga_andamento_processo* que apresenta cento e oitenta e sete (187) tuplas, diferente das demais tabelas que apresentam cento e trinta (130). Isso se justifica pelo fato de que para cada processo cadastrado foi possível o registro de até três (3) estados de andamento do processo, a saber: *PROCESSO CADASTRADO, PROCESSO PROTOCOLIZADO e PROCESSO DISTRIBUÍDO*.

Similaridade entre Tuplas

Uma vez que foi constatado que as réplicas apresentam o mesmo número de registros, se faz necessário verificar se esses registros são consistentes entre as réplicas. Assim a próxima consulta tem o propósito de comparar os registros entre as tabelas, verificando se para cada registro encontrado nas tabelas da réplica 1 existe um e somente um registro nas tabelas correspondentes nas réplicas 2 e 3. Para esta consulta foram criados *links* entre as réplicas do banco de dados, conforme Figura 5.3.

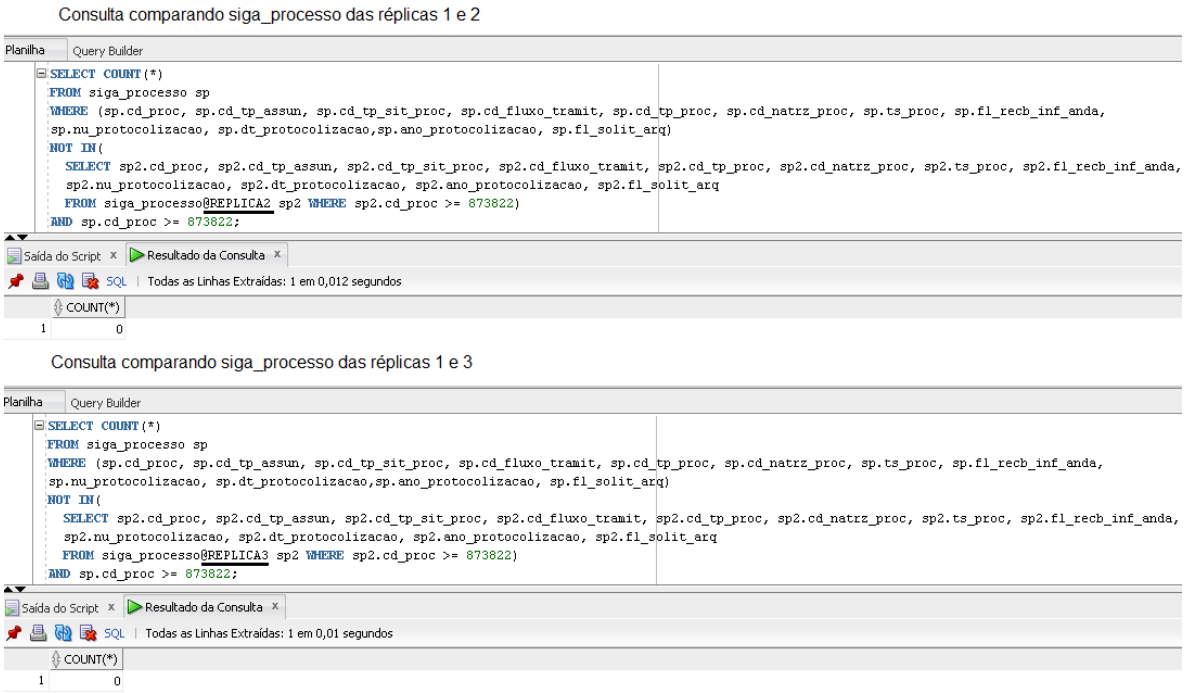
Figura 5.3 – Criação de Links entre as réplicas do banco de dados.

```
--DatabaseLink para a réplica hospedada no NTI/UFPE (150.161.0.33)
CREATE DATABASE LINK "REPLICA2"
CONNECT TO "RIVALDO" IDENTIFIED BY VALUES '05A404A45A0ABEEEADABC94F39C211BC97'
USING '
    (DESCRIPTION =
        (ADDRESS_LIST =
            (ADDRESS = (PROTOCOL = TCP)(HOST = 150.161.0.33)(PORT = 1521))
        )
        (CONNECT_DATA =
        (SERVICE_NAME = dbsiga)
    ))';

--DatabaseLink para a réplica hospedada no CIn/UFPE (172.21.6.116)
CREATE DATABASE LINK "REPLICA3"
CONNECT TO "RIVALDO" IDENTIFIED BY VALUES '05A404A45A0ABEEEADABC94F39C211BC97'
USING '
    (DESCRIPTION =
        (ADDRESS_LIST =
            (ADDRESS = (PROTOCOL = TCP)(HOST = 172.21.6.116)(PORT = 1521))
        )
        (CONNECT_DATA =
        (SERVICE_NAME = dbsiga)
    ))';
```

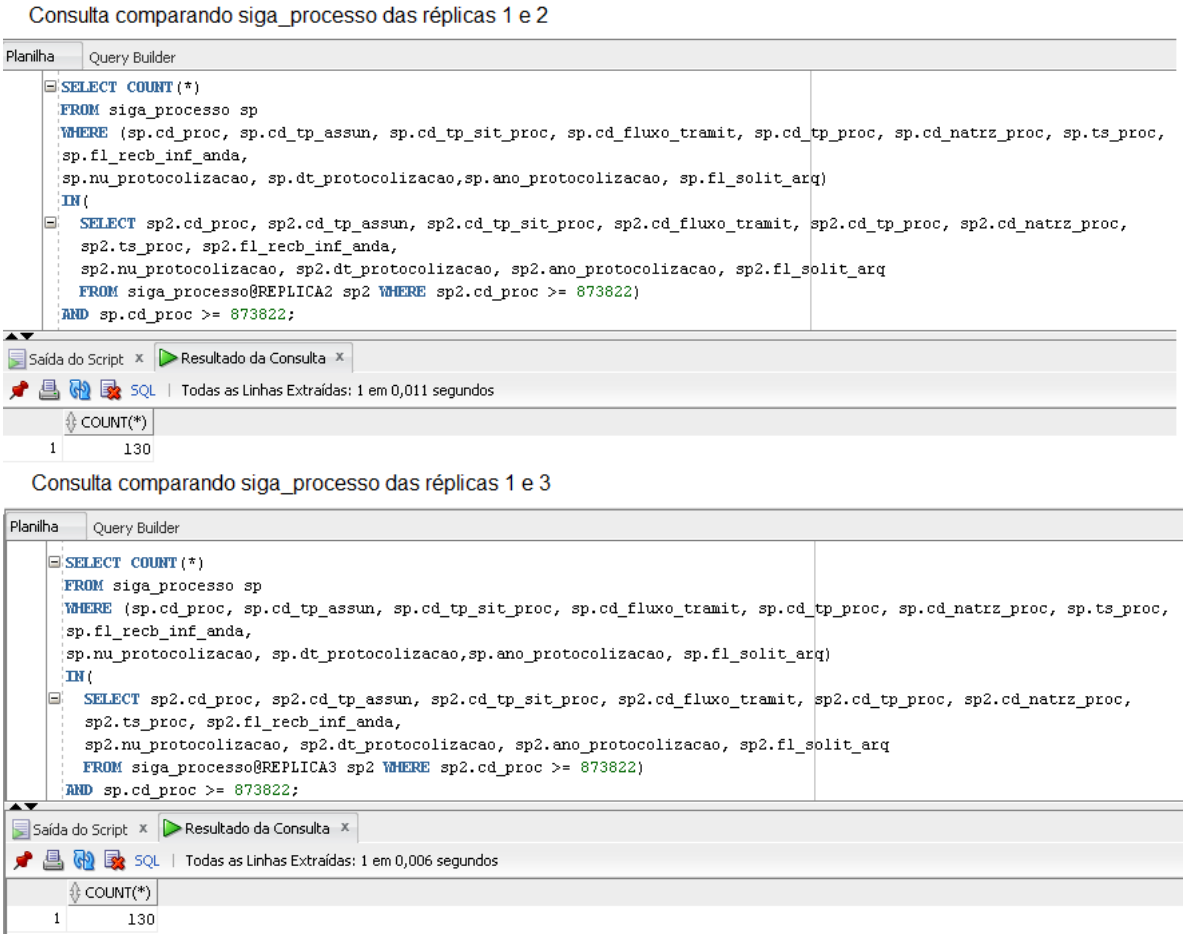
Fonte: Elaborada pelo Autor (2017).

Figura 5.4 – Consulta para verificar se existem registros na tabela siga_processo de uma réplica que não exista na tabela correspondente em outra réplica



Fonte: Elaborada pelo Autor (2017).

Figura 5.5 – Consulta para verificar quantos registros existem na tabela siga_processo de uma réplica que exista na tabela correspondente em outra réplica



Fonte: Elaborada pelo Autor (2017).

Como pode ser observado nas Figuras 5.4 e 5.5 não existe registro na tabela *siga_processo* da réplica 1 que não tenha um correspondente na mesma tabela nas réplicas 2 e 3 (Figura 5.4), e que para cada registro em *siga_processo* da réplica 1 existe apenas um correspondente na mesma tabela nas réplicas 2 e 3.

A mesma consulta foi realizada envolvendo as demais tabelas apresentando o mesmo resultado. Portanto, levando em consideração que as tabelas apresentaram o mesmo número de registros nas três réplicas e que para cada registro numa réplica é encontrado um e apenas um correspondente nas outras

duas réplicas, pode-se afirmar que, ao término dos cadastros, as réplicas apresentaram um estado de consistência de 100%.

2ª Etapa

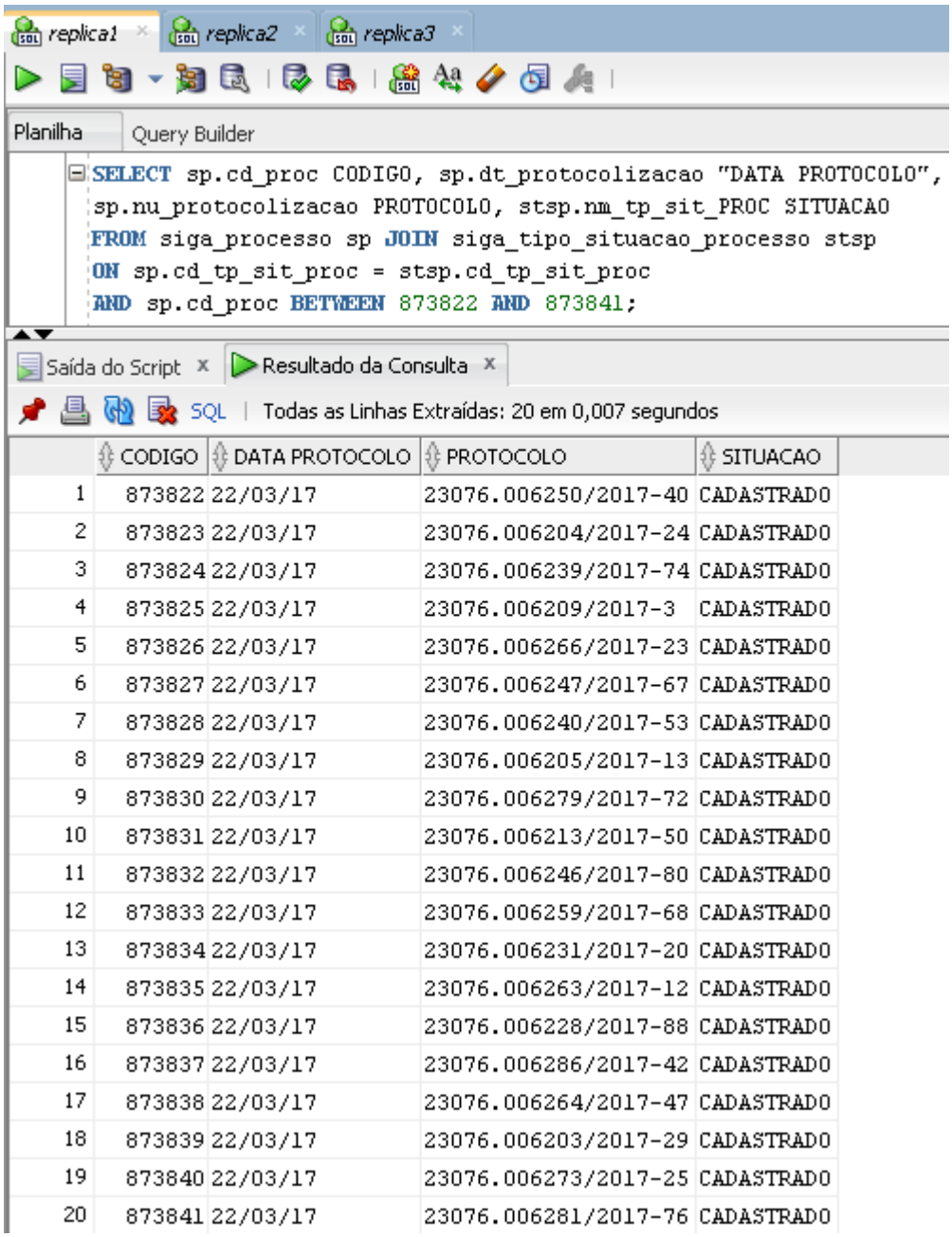
Consultas realizadas após realização de atualizações nos registros, para comprovação que os dados se mantiveram consistentes.

A partir deste ponto foi utilizado um subconjunto dos dados, composto pelos vinte (20) primeiros processos registrados. Essa seleção foi necessária para tornar possível a exibição das consultas em imagens.

A consulta seguinte (Figuras 5.6, 5.7 e 5.8) foi realizada nas tabelas *siga_processo* e *siga_tipo_situacao_processo* (tabela auxiliar que armazena as situações possíveis dos processos) das replicas 1, 2 e 3 respetivamente. Essas tabelas mantém uma relação de N...1 de *siga_processo* para *siga_tipo_situacao_processo*.

Consulta de tuplas antes da aplicação de atualizações

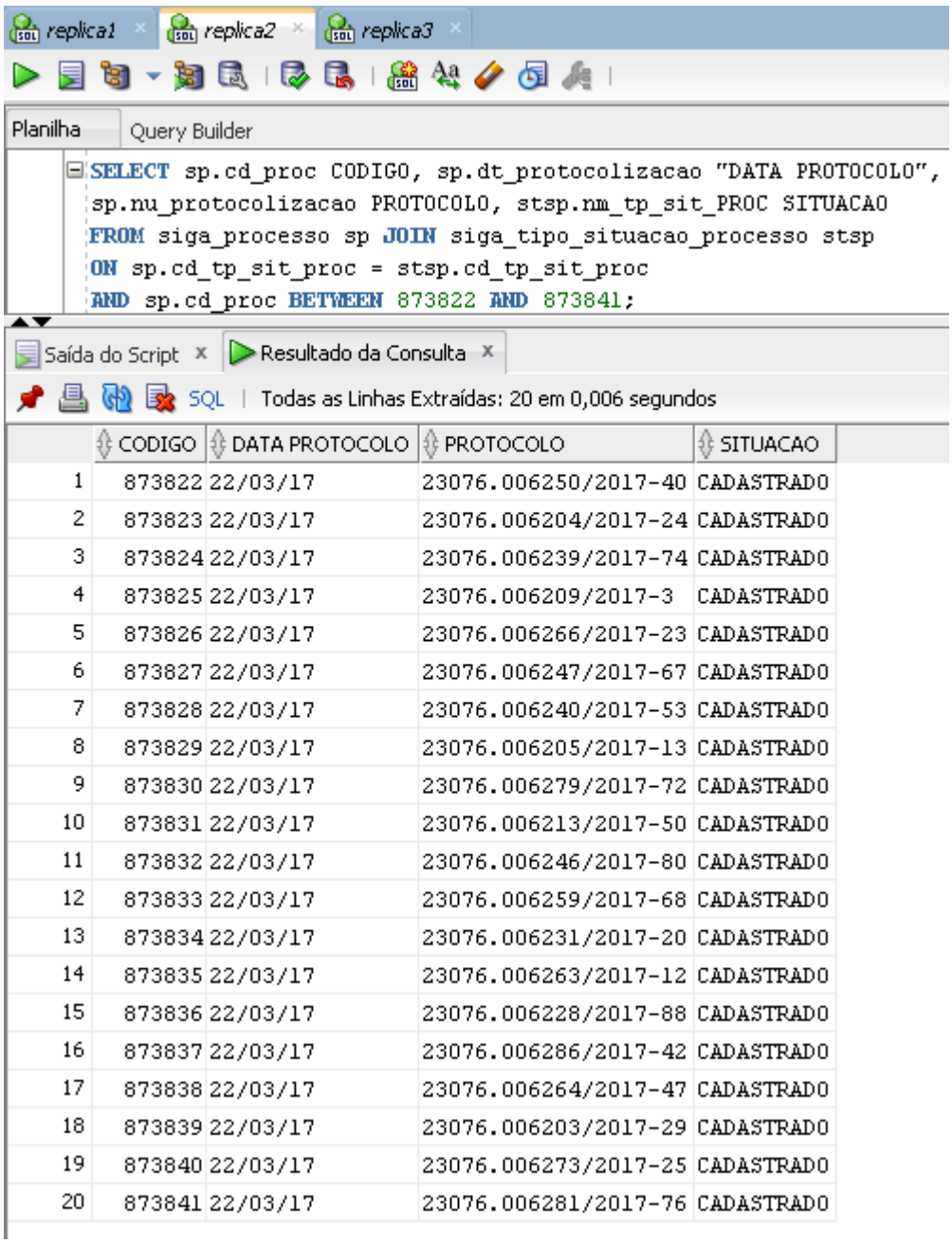
Figura 5.6 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 1 NTI/UFPE.



	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	CADASTRADO
2	873823	22/03/17	23076.006204/2017-24	CADASTRADO
3	873824	22/03/17	23076.006239/2017-74	CADASTRADO
4	873825	22/03/17	23076.006209/2017-3	CADASTRADO
5	873826	22/03/17	23076.006266/2017-23	CADASTRADO
6	873827	22/03/17	23076.006247/2017-67	CADASTRADO
7	873828	22/03/17	23076.006240/2017-53	CADASTRADO
8	873829	22/03/17	23076.006205/2017-13	CADASTRADO
9	873830	22/03/17	23076.006279/2017-72	CADASTRADO
10	873831	22/03/17	23076.006213/2017-50	CADASTRADO
11	873832	22/03/17	23076.006246/2017-80	CADASTRADO
12	873833	22/03/17	23076.006259/2017-68	CADASTRADO
13	873834	22/03/17	23076.006231/2017-20	CADASTRADO
14	873835	22/03/17	23076.006263/2017-12	CADASTRADO
15	873836	22/03/17	23076.006228/2017-88	CADASTRADO
16	873837	22/03/17	23076.006286/2017-42	CADASTRADO
17	873838	22/03/17	23076.006264/2017-47	CADASTRADO
18	873839	22/03/17	23076.006203/2017-29	CADASTRADO
19	873840	22/03/17	23076.006273/2017-25	CADASTRADO
20	873841	22/03/17	23076.006281/2017-76	CADASTRADO

Fonte: Elaborada pelo Autor (2017).

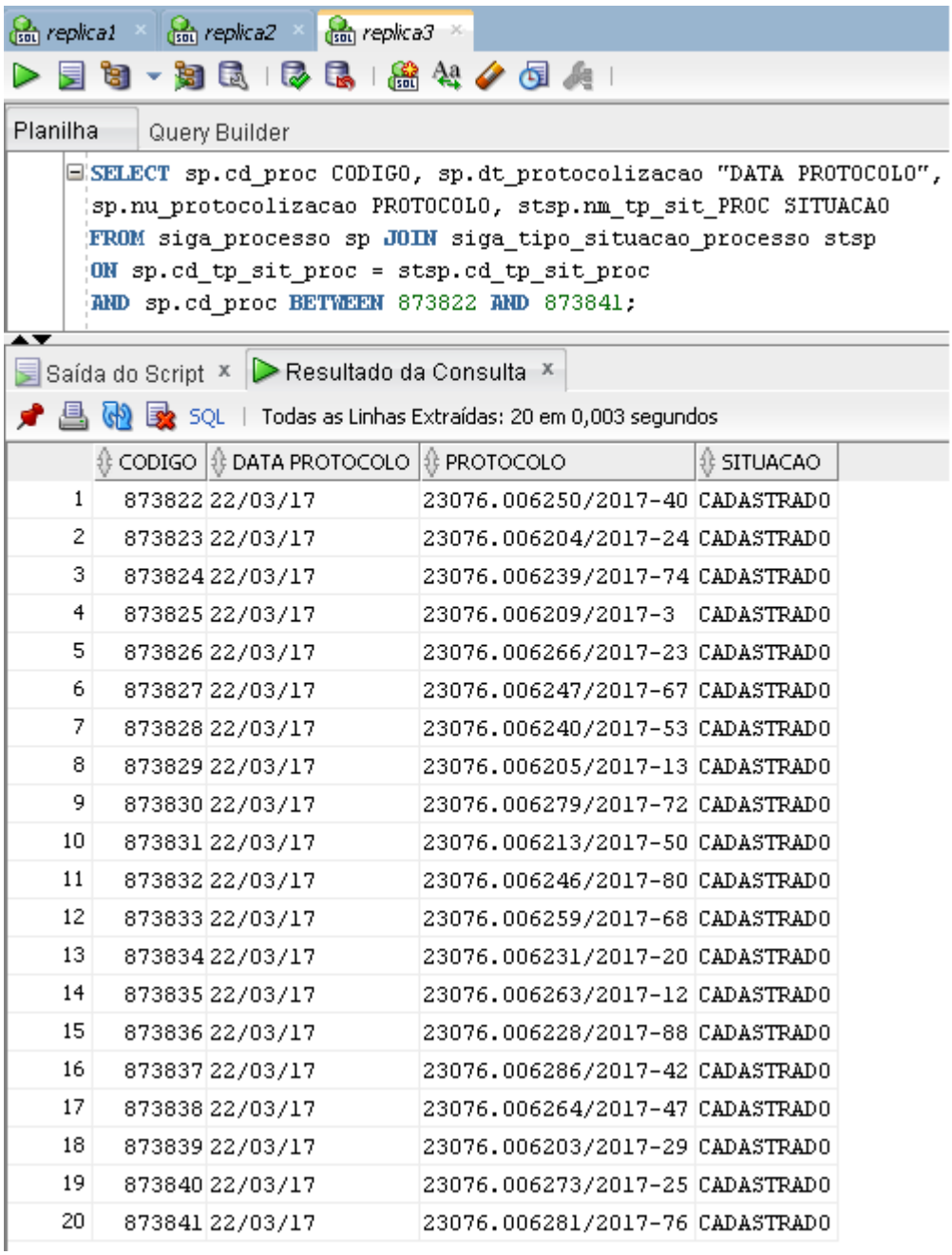
Figura 5.7 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 2 NTI/UFPE.



	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	CADASTRADO
2	873823	22/03/17	23076.006204/2017-24	CADASTRADO
3	873824	22/03/17	23076.006239/2017-74	CADASTRADO
4	873825	22/03/17	23076.006209/2017-3	CADASTRADO
5	873826	22/03/17	23076.006266/2017-23	CADASTRADO
6	873827	22/03/17	23076.006247/2017-67	CADASTRADO
7	873828	22/03/17	23076.006240/2017-53	CADASTRADO
8	873829	22/03/17	23076.006205/2017-13	CADASTRADO
9	873830	22/03/17	23076.006279/2017-72	CADASTRADO
10	873831	22/03/17	23076.006213/2017-50	CADASTRADO
11	873832	22/03/17	23076.006246/2017-80	CADASTRADO
12	873833	22/03/17	23076.006259/2017-68	CADASTRADO
13	873834	22/03/17	23076.006231/2017-20	CADASTRADO
14	873835	22/03/17	23076.006263/2017-12	CADASTRADO
15	873836	22/03/17	23076.006228/2017-88	CADASTRADO
16	873837	22/03/17	23076.006286/2017-42	CADASTRADO
17	873838	22/03/17	23076.006264/2017-47	CADASTRADO
18	873839	22/03/17	23076.006203/2017-29	CADASTRADO
19	873840	22/03/17	23076.006273/2017-25	CADASTRADO
20	873841	22/03/17	23076.006281/2017-76	CADASTRADO

Fonte: Elaborada pelo Autor (2017).

Figura 5.8 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 3 CIn/UFPE.



	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	CADASTRADO
2	873823	22/03/17	23076.006204/2017-24	CADASTRADO
3	873824	22/03/17	23076.006239/2017-74	CADASTRADO
4	873825	22/03/17	23076.006209/2017-3	CADASTRADO
5	873826	22/03/17	23076.006266/2017-23	CADASTRADO
6	873827	22/03/17	23076.006247/2017-67	CADASTRADO
7	873828	22/03/17	23076.006240/2017-53	CADASTRADO
8	873829	22/03/17	23076.006205/2017-13	CADASTRADO
9	873830	22/03/17	23076.006279/2017-72	CADASTRADO
10	873831	22/03/17	23076.006213/2017-50	CADASTRADO
11	873832	22/03/17	23076.006246/2017-80	CADASTRADO
12	873833	22/03/17	23076.006259/2017-68	CADASTRADO
13	873834	22/03/17	23076.006231/2017-20	CADASTRADO
14	873835	22/03/17	23076.006263/2017-12	CADASTRADO
15	873836	22/03/17	23076.006228/2017-88	CADASTRADO
16	873837	22/03/17	23076.006286/2017-42	CADASTRADO
17	873838	22/03/17	23076.006264/2017-47	CADASTRADO
18	873839	22/03/17	23076.006203/2017-29	CADASTRADO
19	873840	22/03/17	23076.006273/2017-25	CADASTRADO
20	873841	22/03/17	23076.006281/2017-76	CADASTRADO

Fonte: Elaborada pelo Autor (2017).

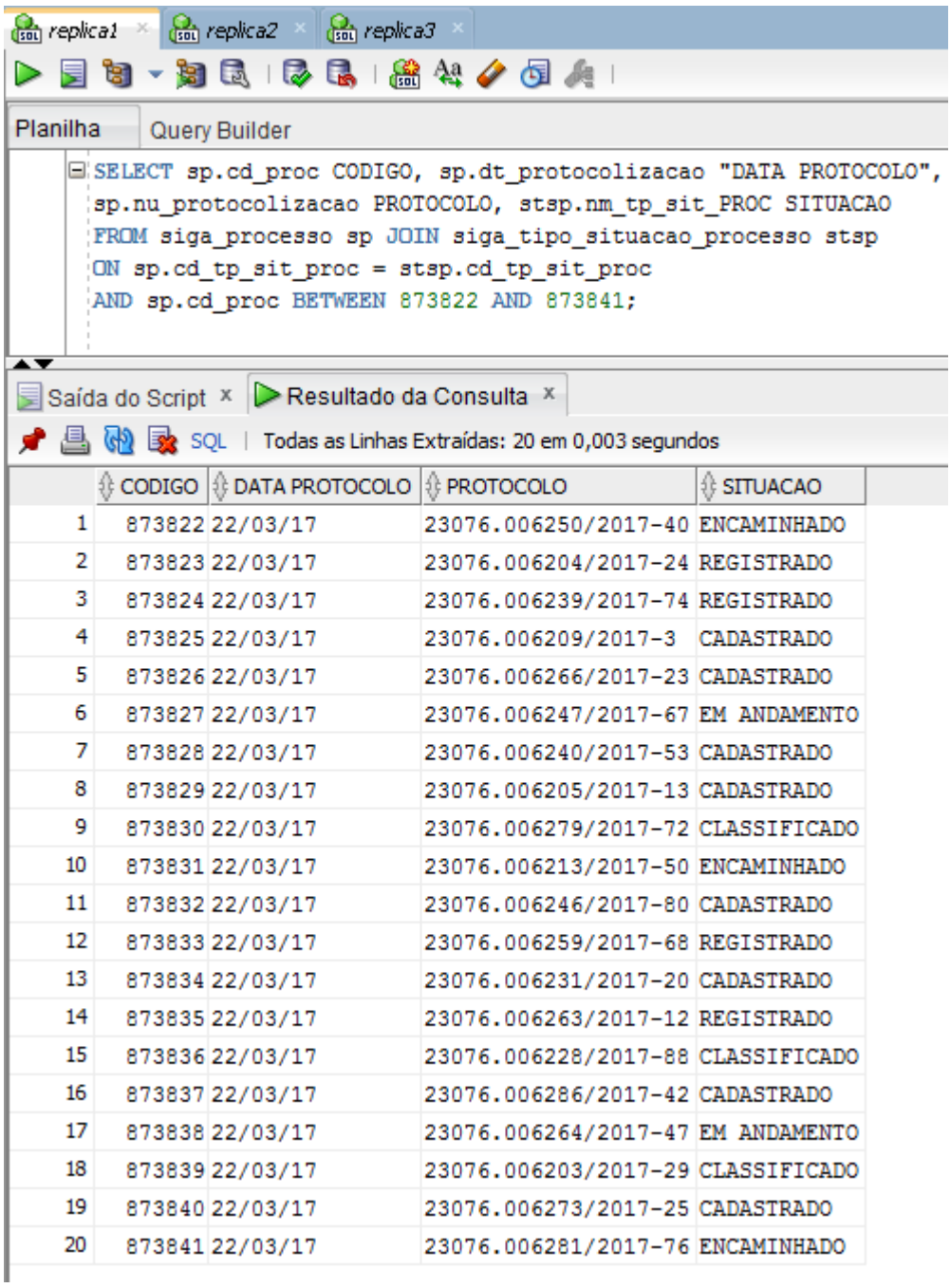
Além da consistência, pode-se observar que todos os processos encontram-se com a situação de 'CADASTRADO'. Destes vinte (20) processos selecionados para testes de atualizações, doze (12) foram atualizados para situações diferentes e a consulta foi refeita para avaliar o resultados nas réplicas.

As atualizações foram executadas por meio de alterações no *sigaProcesso*, da mesma forma que os registros. No entanto, para realizar as atualizações foi necessário apenas um operador (o pesquisador).

Consulta de tuplas após aplicação de atualizações

Após as atualizações serem executadas nas réplicas por meio do *middleware*, as consultas anteriores foram refeitas afim de avaliar o resultado dessas atualizações.

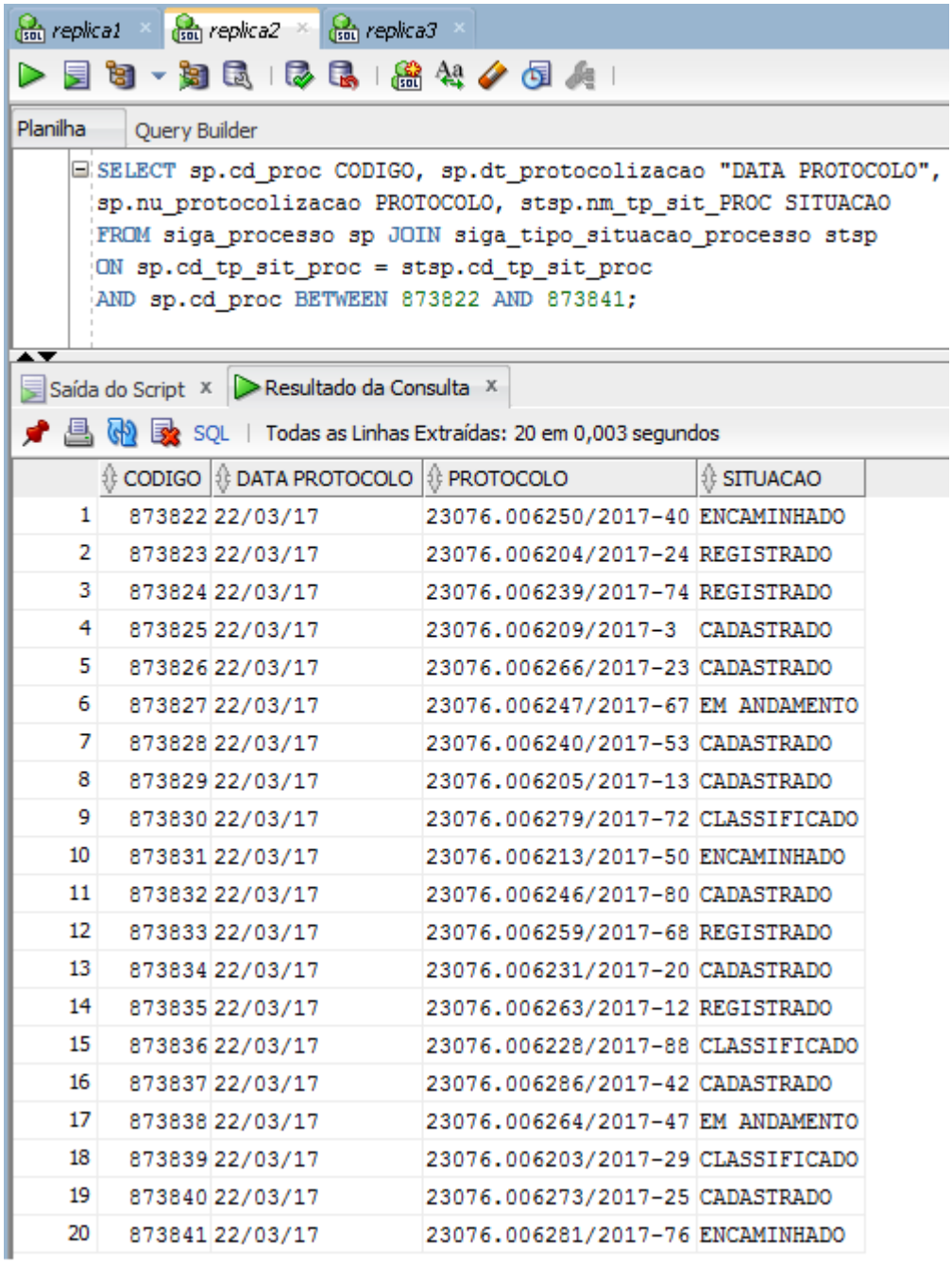
Figura 5.9 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 1 NTI/UFPE.



	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	ENCAMINHADO
2	873823	22/03/17	23076.006204/2017-24	REGISTRADO
3	873824	22/03/17	23076.006239/2017-74	REGISTRADO
4	873825	22/03/17	23076.006209/2017-3	CADASTRADO
5	873826	22/03/17	23076.006266/2017-23	CADASTRADO
6	873827	22/03/17	23076.006247/2017-67	EM ANDAMENTO
7	873828	22/03/17	23076.006240/2017-53	CADASTRADO
8	873829	22/03/17	23076.006205/2017-13	CADASTRADO
9	873830	22/03/17	23076.006279/2017-72	CLASSIFICADO
10	873831	22/03/17	23076.006213/2017-50	ENCAMINHADO
11	873832	22/03/17	23076.006246/2017-80	CADASTRADO
12	873833	22/03/17	23076.006259/2017-68	REGISTRADO
13	873834	22/03/17	23076.006231/2017-20	CADASTRADO
14	873835	22/03/17	23076.006263/2017-12	REGISTRADO
15	873836	22/03/17	23076.006228/2017-88	CLASSIFICADO
16	873837	22/03/17	23076.006286/2017-42	CADASTRADO
17	873838	22/03/17	23076.006264/2017-47	EM ANDAMENTO
18	873839	22/03/17	23076.006203/2017-29	CLASSIFICADO
19	873840	22/03/17	23076.006273/2017-25	CADASTRADO
20	873841	22/03/17	23076.006281/2017-76	ENCAMINHADO

Fonte: Elaborada pelo Autor (2017).

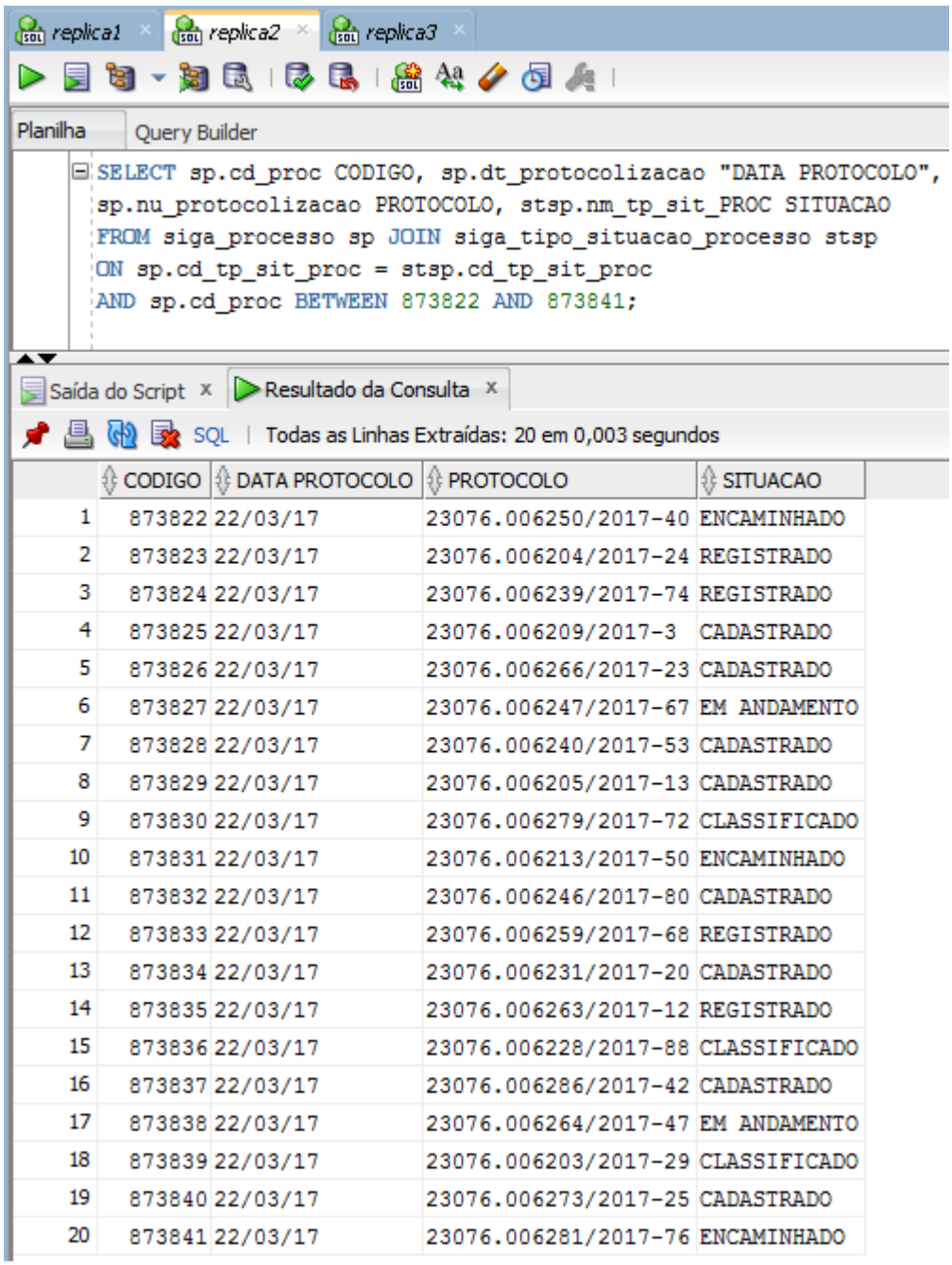
Figura 5.10 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 2 NTI/UFPE.



	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	ENCAMINHADO
2	873823	22/03/17	23076.006204/2017-24	REGISTRADO
3	873824	22/03/17	23076.006239/2017-74	REGISTRADO
4	873825	22/03/17	23076.006209/2017-3	CADASTRADO
5	873826	22/03/17	23076.006266/2017-23	CADASTRADO
6	873827	22/03/17	23076.006247/2017-67	EM ANDAMENTO
7	873828	22/03/17	23076.006240/2017-53	CADASTRADO
8	873829	22/03/17	23076.006205/2017-13	CADASTRADO
9	873830	22/03/17	23076.006279/2017-72	CLASSIFICADO
10	873831	22/03/17	23076.006213/2017-50	ENCAMINHADO
11	873832	22/03/17	23076.006246/2017-80	CADASTRADO
12	873833	22/03/17	23076.006259/2017-68	REGISTRADO
13	873834	22/03/17	23076.006231/2017-20	CADASTRADO
14	873835	22/03/17	23076.006263/2017-12	REGISTRADO
15	873836	22/03/17	23076.006228/2017-88	CLASSIFICADO
16	873837	22/03/17	23076.006286/2017-42	CADASTRADO
17	873838	22/03/17	23076.006264/2017-47	EM ANDAMENTO
18	873839	22/03/17	23076.006203/2017-29	CLASSIFICADO
19	873840	22/03/17	23076.006273/2017-25	CADASTRADO
20	873841	22/03/17	23076.006281/2017-76	ENCAMINHADO

Fonte: Elaborada pelo Autor (2017).

Figura 5.11 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 3 CIn/UFPE.



	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	ENCAMINHADO
2	873823	22/03/17	23076.006204/2017-24	REGISTRADO
3	873824	22/03/17	23076.006239/2017-74	REGISTRADO
4	873825	22/03/17	23076.006209/2017-3	CADASTRADO
5	873826	22/03/17	23076.006266/2017-23	CADASTRADO
6	873827	22/03/17	23076.006247/2017-67	EM ANDAMENTO
7	873828	22/03/17	23076.006240/2017-53	CADASTRADO
8	873829	22/03/17	23076.006205/2017-13	CADASTRADO
9	873830	22/03/17	23076.006279/2017-72	CLASSIFICADO
10	873831	22/03/17	23076.006213/2017-50	ENCAMINHADO
11	873832	22/03/17	23076.006246/2017-80	CADASTRADO
12	873833	22/03/17	23076.006259/2017-68	REGISTRADO
13	873834	22/03/17	23076.006231/2017-20	CADASTRADO
14	873835	22/03/17	23076.006263/2017-12	REGISTRADO
15	873836	22/03/17	23076.006228/2017-88	CLASSIFICADO
16	873837	22/03/17	23076.006286/2017-42	CADASTRADO
17	873838	22/03/17	23076.006264/2017-47	EM ANDAMENTO
18	873839	22/03/17	23076.006203/2017-29	CLASSIFICADO
19	873840	22/03/17	23076.006273/2017-25	CADASTRADO
20	873841	22/03/17	23076.006281/2017-76	ENCAMINHADO

Fonte: Elaborada pelo Autor (2017).

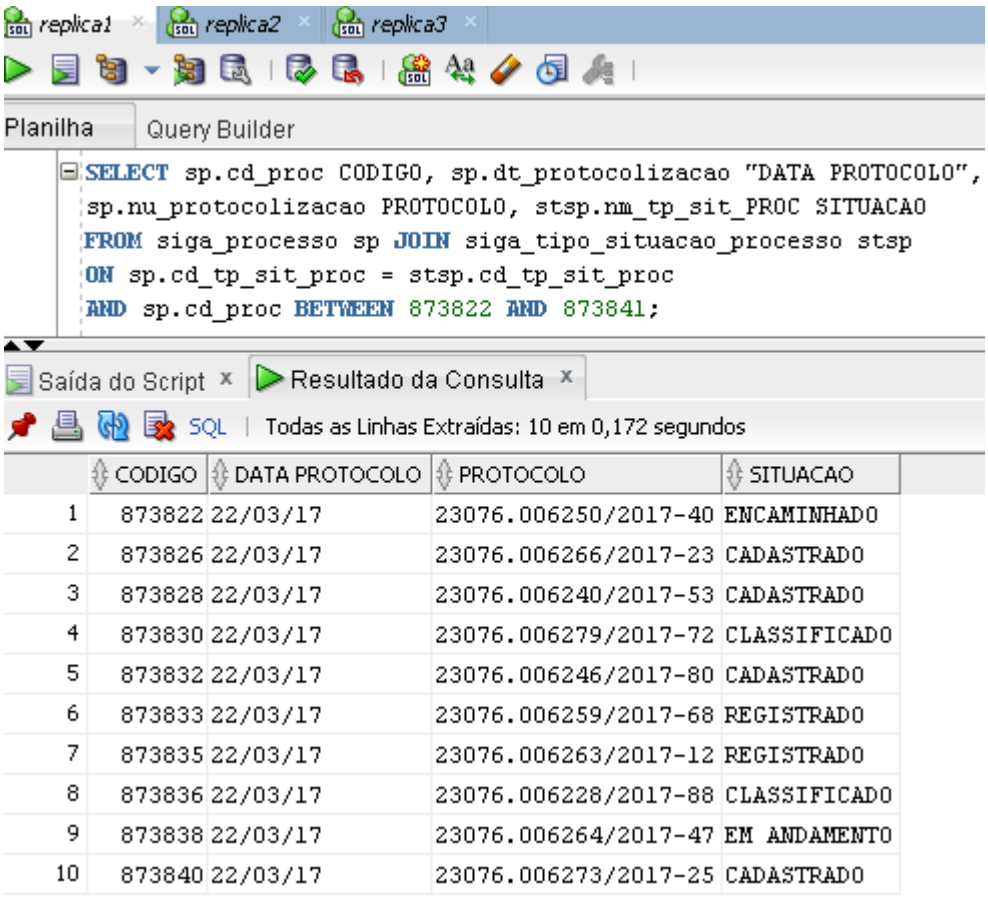
Os estados das tabelas *siga_processo* verificados nas Figuras 5.9, 5.10 e 5.11 comprovam que, após as atualizações, as mesmas se mantiveram consistentes.

3ª Etapa

Seguindo a programação dos testes foram realizadas exclusões de dez (10) registros, dentre os selecionados para os testes de atualizações. Este procedimento foi realizado via aplicação *sigaProcesso* sendo realizado por um operador, como no teste anterior. As relações mantidas entre as tabelas *siga_tramitacao_processo, siga_solicitante* e *siga_andamento_processo* com a tabela *siga_processo* tem a clausula ON DELETE CASCADE, onde ao excluir um registro em siga_processo o registro correspondente das demais tabelas são excluídos automaticamente. Sendo assim as exclusões foram realizadas apenas em *siga_processo*.

As Figuras 5.9, 5.10 e 5.11 devem ser consideradas como o estado das tabelas *siga_processo* nas réplicas antes das exclusões. Realizadas as exclusões, seu resultado pode ser verificado nas Figuras 5.12, 5.13 e 5.14.

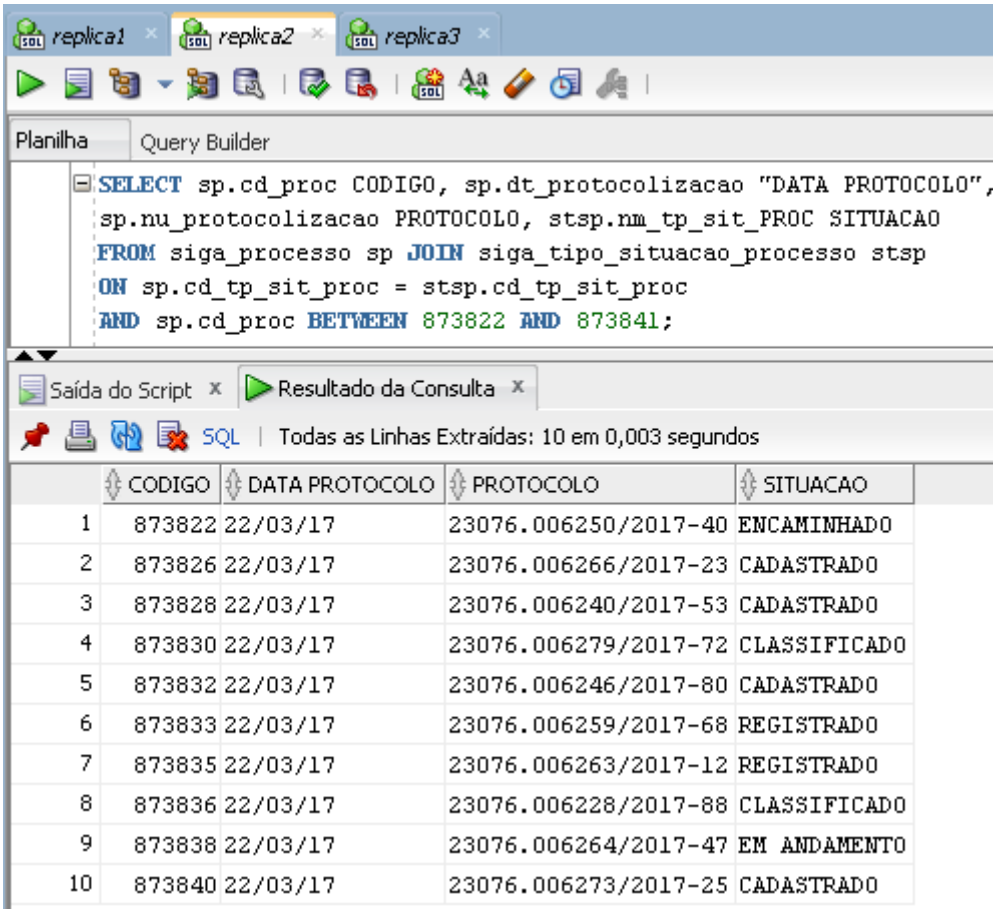
Figura 5.12 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 1 NTI/UFPE.



	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	ENCAMINHADO
2	873826	22/03/17	23076.006266/2017-23	CADASTRADO
3	873828	22/03/17	23076.006240/2017-53	CADASTRADO
4	873830	22/03/17	23076.006279/2017-72	CLASSIFICADO
5	873832	22/03/17	23076.006246/2017-80	CADASTRADO
6	873833	22/03/17	23076.006259/2017-68	REGISTRADO
7	873835	22/03/17	23076.006263/2017-12	REGISTRADO
8	873836	22/03/17	23076.006228/2017-88	CLASSIFICADO
9	873838	22/03/17	23076.006264/2017-47	EM ANDAMENTO
10	873840	22/03/17	23076.006273/2017-25	CADASTRADO

Fonte: Elaborada pelo Autor (2017).

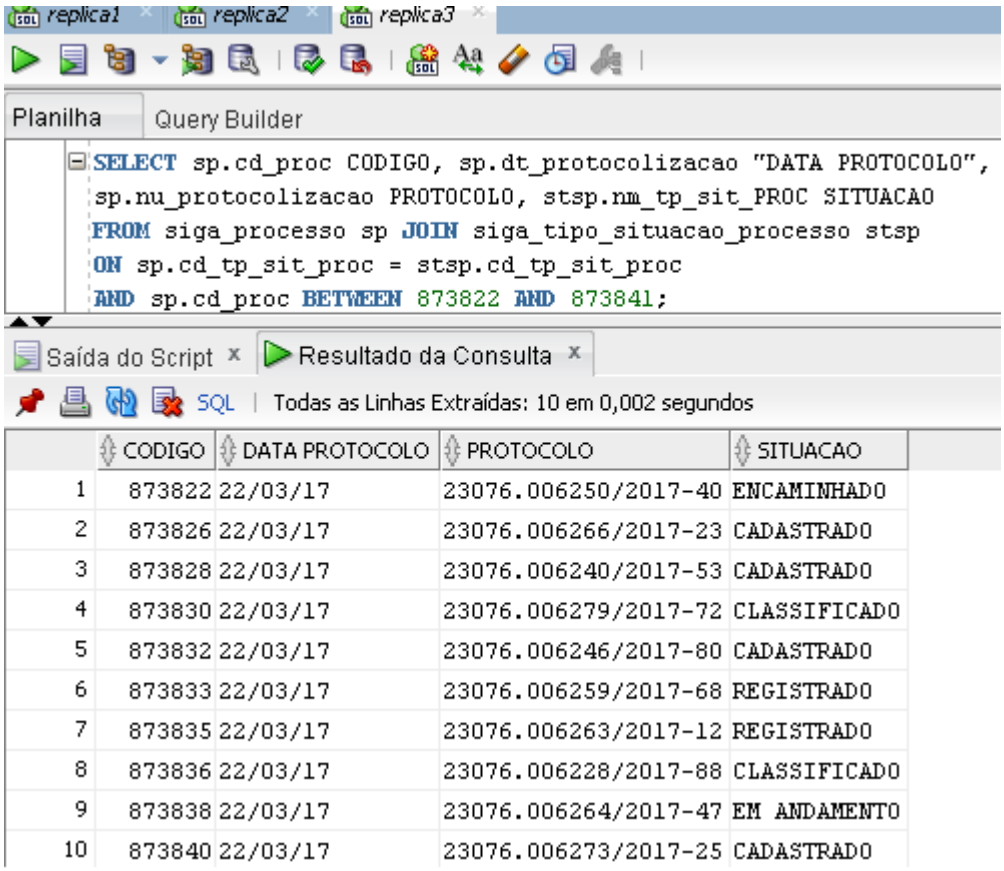
Figura 5.13 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 2 NTI/UFPE.



	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	ENCAMINHADO
2	873826	22/03/17	23076.006266/2017-23	CADASTRADO
3	873828	22/03/17	23076.006240/2017-53	CADASTRADO
4	873830	22/03/17	23076.006279/2017-72	CLASSIFICADO
5	873832	22/03/17	23076.006246/2017-80	CADASTRADO
6	873833	22/03/17	23076.006259/2017-68	REGISTRADO
7	873835	22/03/17	23076.006263/2017-12	REGISTRADO
8	873836	22/03/17	23076.006228/2017-88	CLASSIFICADO
9	873838	22/03/17	23076.006264/2017-47	EM ANDAMENTO
10	873840	22/03/17	23076.006273/2017-25	CADASTRADO

Fonte: Elaborada pelo Autor (2017).

Figura 5.14 – Consulta em siga_processo e siga_tipo_sit_processo na réplica 3 CIn/UFPE.

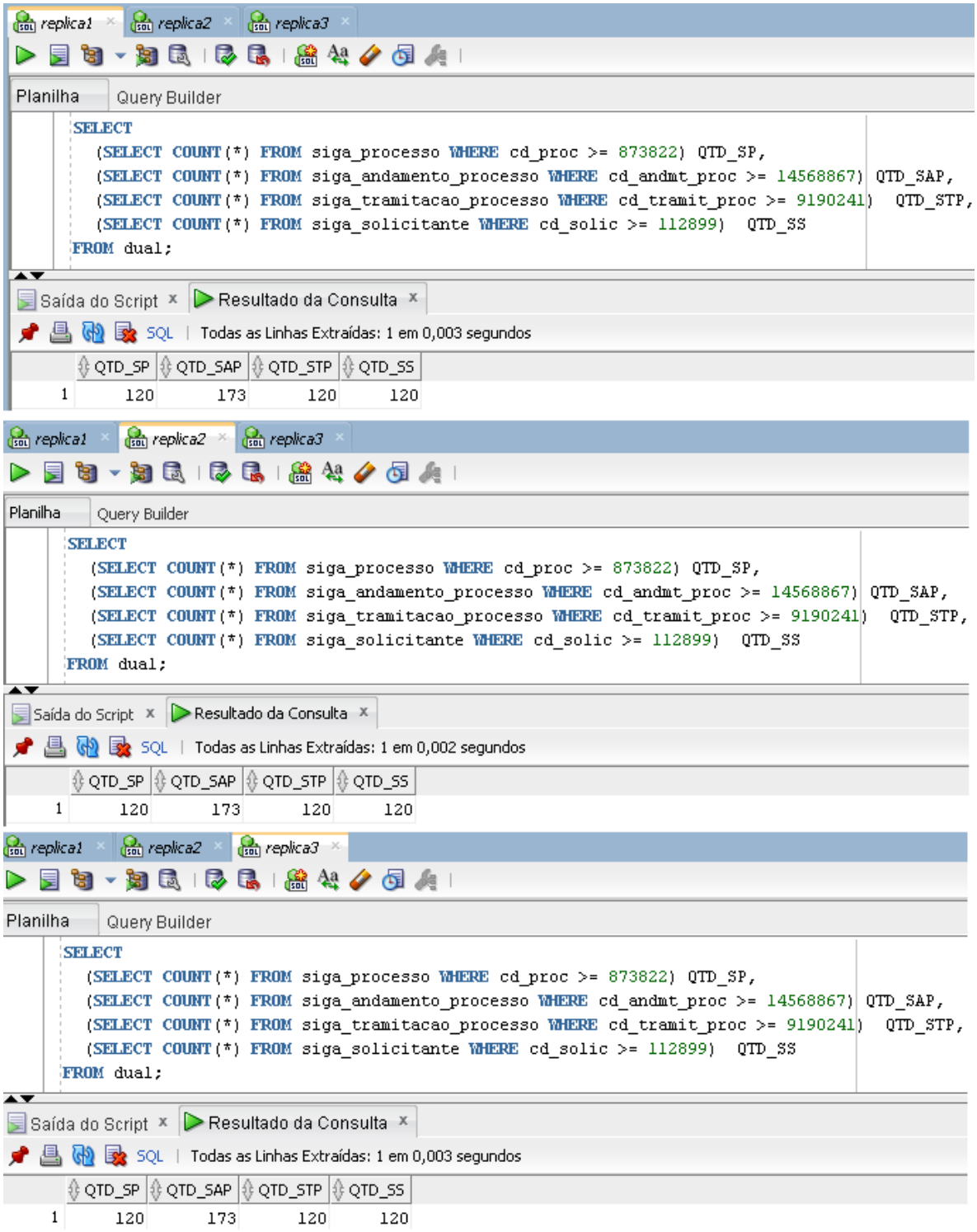


	CODIGO	DATA PROTOCOLO	PROTOCOLO	SITUACAO
1	873822	22/03/17	23076.006250/2017-40	ENCAMINHADO
2	873826	22/03/17	23076.006266/2017-23	CADASTRADO
3	873828	22/03/17	23076.006240/2017-53	CADASTRADO
4	873830	22/03/17	23076.006279/2017-72	CLASSIFICADO
5	873832	22/03/17	23076.006246/2017-80	CADASTRADO
6	873833	22/03/17	23076.006259/2017-68	REGISTRADO
7	873835	22/03/17	23076.006263/2017-12	REGISTRADO
8	873836	22/03/17	23076.006228/2017-88	CLASSIFICADO
9	873838	22/03/17	23076.006264/2017-47	EM ANDAMENTO
10	873840	22/03/17	23076.006273/2017-25	CADASTRADO

Fonte: Elaborada pelo Autor (2017).

Como pode ser observado nas Figuras 5.12, 5.13 e 5.14 as tabelas *siga_processo* nas réplicas se mantiveram consistentes entre si, apresentando um mesmo estado após a exclusão de dez (10) tuplas. Para confirmar que nas demais tabelas envolvidas essa a consistência foi mantida, as consultas da Figura 5.2 foram novamente realizadas.

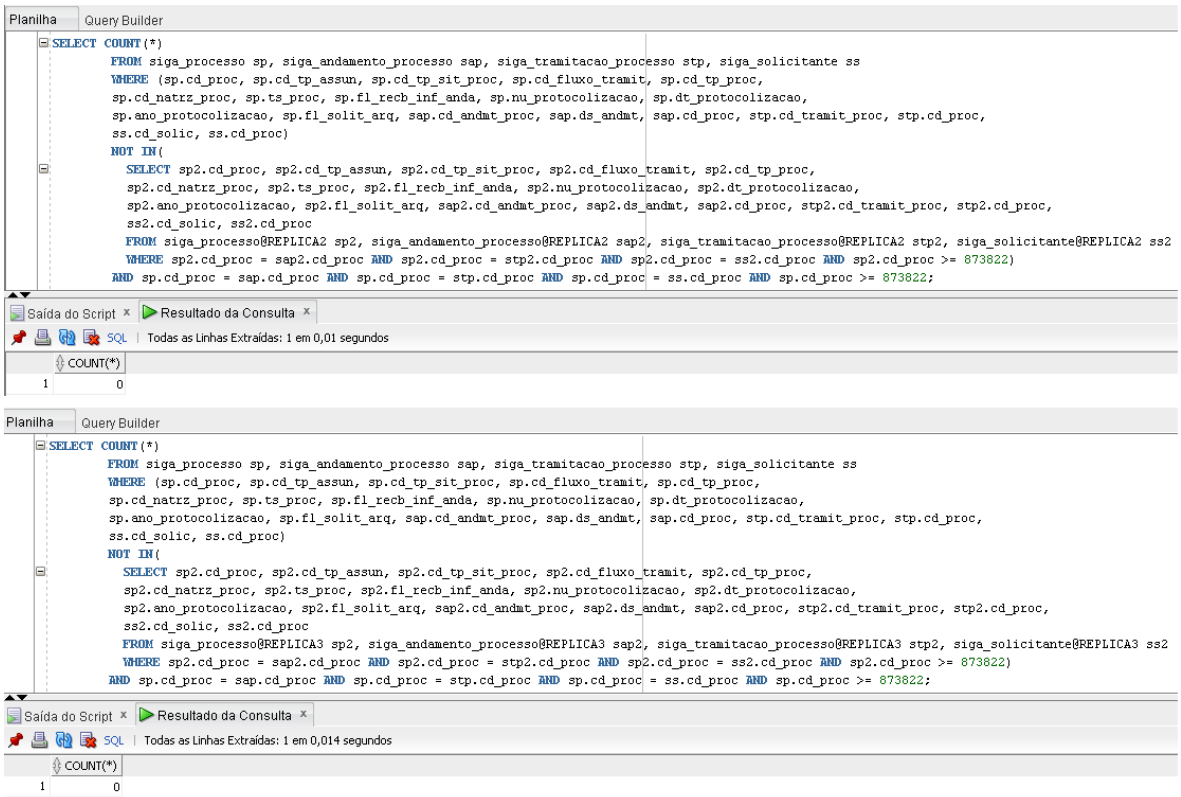
Figura 5.15 – Consulta para contabilizar registros nas tabelas após exclusão.



Fonte: Elaborada pelo Autor (2017).

Como pode-se verificar na Figura 5.15 as tuplas foram excluídas em todas tabelas envolvidas nas três réplicas. Para finalizar a consulta da Figura 5.16 fez uma junção das quatro tabelas envolvidas afim de verificar registros presentes em cada réplica, comprovando se as réplicas se mantiveram consistentes após as três operações realizadas, cadastro, atualização e exclusão.

Figura 5.16 – Consulta comparativa das tabelas entre as réplicas.



Fonte: Elaborada pelo Autor (2017).

As consultas da Figura 5.16 comprovam que não existe um registro do conjunto formado pela junção das quatro (4) tabelas que exista na réplica 1 e não exista nas réplicas 2 e 3.

5.3 Conclusões do Capítulo

Este capitulo especificou o ambiente criado com o objetivo de simular uma situação real e apresentou os testes realizados no protótipo e seus resultados:

Primeiro: foi realizada uma carga de dados através do *sigaProcesso* e dos middleware, após essa carga foram feitos testes, com consultas, com comprovaram que as réplicas se mantiveram consistente após o cadastro dos processos.

Segundo: foi selecionado um sub conjunto dos processos com vinte (20) registros e nesse sub conjunto foram realizadas, através do *sigaProcesso*, atualizações na tabela *siga_processo*, após esse procedimento foram realizadas consultas que confirmaram a manutenção da consitência.

Terceiro: dez (10) registros foram excluídos do subconjunto selecionado no teste anterior, feito realizado outra vez através do *sigaProcesso*, após isso consultas foram realizadas verificando que a consistência foi mantida.

O capítulo seguinte deste trabalho apresenta as conclusões baseadas nos resultados dos testes realizados, bem como aborda as contribuições e limitações, além de propor trabalhos futuros.

6 CONSIDERAÇÕES FINAIS

Este capítulo tem por objetivo apresentar as considerações finais deste trabalho, enfatizando suas principais contribuições e as limitações do modelo, além da sugestão de trabalhos futuros.

6.1 Contribuições

A principal contribuição deste trabalho é a utilização da abordagem de atualização ansiosa para auxiliar na manutenção da consistência em bancos de dados relacionais distribuídos.

Outra contribuição é o desenvolvimento de um protótipo que implementa o modelo, o que possibilitou a realização de testes no mesmo. A utilização do Java *Message Service* para a propagação de dados é outro ponto forte deste trabalho, pois o mesmo trouxe segurança e confiabilidade ao modelo, uma vez que garante a entrega dos dados sem repetição ou falhas.

Apesar dos testes terem sido realizados no SGBD Oracle (utilizado pelo sigaProcesso, sistema escolhido como estudo de caso), o protótipo pode ser utilizado com outros SGBD, ajustando-se apenas os dados da conexão e a criação de comandos SQL, caso seja necessário, pois o protótipo aqui construído não depende em nada do SGBD utilizado.

Em razão dos resultados dos testes realizados, descritos no Capítulo 5, conclui-se que o objetivo estabelecido por este trabalho "desenvolver um Middleware Orientado a Mensagem que garanta aplicação de comandos *insert*, *delete* e *update*, assincronamente em réplicas de banco de dados" foi alcançado.

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

6.2 Limitações

A principal limitação deste trabalho encontra-se no ambiente utilizado para o desenvolvimento do protótipo, uma vez que não houve disponibilidade de máquinas distribuídas geograficamente para simular um ambiente mais próximo da realidade enfrentada pelos desenvolvedores quando precisam replicar dados. Desse modo é preciso considerar que os testes realizados não podem ser generalizados de modo a cobrir uma realidade muito distinta da considerada neste trabalho.

O fato de que o protótipo não foi preparado para lidar com situações nas quais uma réplica qualquer, por algum motivo, não consiga consumir uma mensagem (aplicar os comando SQL contido na mensagem), caso as demais réplicas tenham conseguido consumir essa mensagem, apesar de não fazer parte do escopo desta pesquisa, é sem dúvida, uma limitação da solução proposta.

Outra limitação é o fato desta pesquisa não lidar com a latência. Segundo ABADI (2012) quando não há uma partição presente o sistema deve lidar com o conflito entre consistência e latência e ignorar a latência em prol da consistência sem um estudo prévio, é um descuido. Está pesquisa não abordou esse problema, pois a latência presente no ambiente criado era irrelevante.

Apesar da existência destas limitações, elas foram consideradas a fim de minimizar seu impacto sobre o resultado final do trabalho.

Na seção seguinte, são apresentadas as recomendações de trabalhos futuros

6.3 Trabalhos Futuros

Como proposta de trabalhos futuros, recomenda-se a realização de testes em um ambiente mais próximo da realidade, com um maior número de acessos e registros realizados e com réplicas distribuídas geograficamente.

Ademais, as réplicas não atualizadas deveriam ficar indisponíveis para operações até que tenham as atualizações aplicadas. Essa funcionalidade poderia afetar positivamente a manutenção da consistência.

Por fim recomenda-se levar em consideração a latência ao se abordar o problema da consistência, criando assim um modelo mais completo e apto a ser utilizado em ambientes geograficamente distribuídos.

REFERÊNCIAS

ABADI, D. **Consistency tradeoffs in modern distributed database system design: Cap is only part of the story**, Computer, vol. 45, no. 2, pp. 37–42, 2012

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

AGRAWAL D, El ABBADI A., EMEKCI F and METWALLY A. "**Database Management as a Service: Challenges and Opportunities**" Proc. ICDE, pp 1709-1716, 2009

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

CHAVES S.A.; URIARTE R.B, WESTPHALL C.B, **Toward an Architecture for Monitoring Private Clouds.** IEEE Comm. Magazine, vol. 49, no. 12, pp. 130-137, Dec. 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.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. abreviado.**Sistemas Distribuídos: Conceitos e Projeto**. 5ª Edição. Porto Alegre: Bookman, 2013. 1055p.

CRASSO, M, ZUNINO A, CAMPO, M. **A Survey of approaches to web service discovery in service-oriented architectures**. Journal of Database Management (JDM), 22 (2011), pp. 102-132.

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.

FREITAS, E. L. S. X. **KNOWING: UM MODELO PARA GARANTIA DE CONSISTÊNCIA DOS DADOS EM SISTEMAS DE BANCO DE DADOS RELACIONAIS EM NUVEM.** 2014. 106 f. Dissertação (Mestrado em Ciência da Computação) UNIVERSIDADE FEDERAL DE PERNAMBUCO, Pernambudo. 2014.

FREITAS E. L. S. X.; SOUZA. F. F. **Banco de Dados em Nuvem: Um Modelo para Garantia de Consistência dos Dados**. In: Simpósio Brasileiro de Banco de Dados, 2013, Recife. Workshop de Teses e Dissertações, 2013. p. 15-21.

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.

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.

IBRAHIM, N. M, HASSAN, M. F. **Agente-based Message Oriented Middleware (MOM) for cross-platform communication in SOA systems.** *Computer and Information Sciences (ICCIS), 2012 International Conference on*, Kuala Lumpeu, 2012, pp. 547-552.

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: 31/03/2016.

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

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

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

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

W. E. Dong, W. Nan and L. Xu, **QoS-Oriented Monitoring Model of Cloud Computing Resources Availability**, *Computational and Information Sciences (ICCIS), 2013 Fifth International Conference on*, Shiyang, 2013, pp. 1537-1540.

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

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

S. Sakr and A. Liu, **SLA-Based and Consumer-Centric DynamicProvisioning for Cloud Databases**, in Proc. IEEE Cloud, 2012, pp. 360-367.

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.