colloquium exactarum

INTEGRAÇÃO ENTRE BANCO DE DADOS DISTRIBUÍDOS APLICANDO AS TÉCNICAS DE ENTERPRISE INTEGRATIONS PARTNERS

INTEGRATION BETWEEN DISTRIBUTED DATABASES APPLYING ENTERPRISE INTEGRATIONS PARTNERS TECHNICS

Diego Damasceno Pêgo, Francisco Assis da Silva, Francisco Virginio Maracci, Mário Augusto Pazoti

Faculdade de Informática – FIPP, Universidade do Oeste Paulista – UNOESTE
diegodamascenopego@hotmail.com, {chico, francisco, mario}@unoeste.br

RESUMO – Com a atual demanda do setor varejista em expandir suas lojas para locais estratégicos, inicialmente, a estratégia adotada era de manter um banco de dados centralizado, em que o mesmo era responsável por receber requisições de consultas e gravações de novas informações. Porém, alguns fatores como volume de dados, desempenho nas consultas, disponibilidade, concorrência e integridade, ao longo dessa trajetória demonstraram essa opção ineficiente. Como forma de atender a essa demanda e ainda não exigir grandes modificações nos softwares legados, busca-se, neste trabalho, uma solução que descentralize a informação, mantendo os dados mais próximos do local onde serão utilizados. Torna-se necessário desenvolver um mecanismo que possa manter algumas informações sincronizadas entre as bases de dados locais, aplicando técnicas de EIPs (*Enterprise Integration Partners*). Os resultados foram validados em laboratório e baseados em um estudo de caso real de um software para farmácias.
Palavras-chave: Integração; EIPs; Banco de Dados Distribuídos; Descentralização.

ABSTRACT – With the new retail business' strategy of expanding its area and relocating its stores to strategic locations, initially, companies centralized their data into one central database, which was responsible for receiving query requests and record new information. However, some issues as data volume, availability, competition and integrity demonstrate this option no longer efficient. To respond to these new market's demand and need, it's sought a solution that incorporates distributed databases technics, keeping data near to its location use. For this, it's needed to develop a mechanism that keeps information in synch with other databases, applying the EIP's (Enterprise Integration Partners) technics. The results were validated in laboratory and based on a real case study at a company that develops solutions for drugstores.
Keywords: Integration; EIPs; Distributed Databases; Decentralization.

Recebido em: 04/08/2016
Revisado em: 17/05/2017
Aprovado em: 19/10/2017

1. INTRODUÇÃO

De acordo com pesquisa realizada pelo IDV (Instituto de Desenvolvimento do Varejo)[1], o segmento varejista é o setor da economia que mais cresce. Nos últimos 10 anos, apresentou um crescimento três vezes maior que o PIB[2] brasileiro e se destaca como setor que mais gera empregos formais no país (IDV, 2014).

Segundo o IBGE[3] (2015), o setor farmacêutico é um dos principais responsáveis por manter esse crescimento acentuado. Como se trata de um setor em expansão e com rigoroso controle fiscal, cada vez mais as empresas buscam recursos tecnológicos que atendam às suas necessidades em disponibilidade, segurança, integridade, desempenho e escalabilidade.

Por se tratar de um segmento composto por uma estrutura de franquias e filiais, é comum que as empresas de desenvolvimento de software[4] ofereçam a seus clientes uma arquitetura de banco de dados centralizada, podendo existir ou não outros bancos de dados locais.

A informação é fator predominante para o sucesso e para a expansão do setor varejista, enquadrando-se como fonte de estatística para tomada de decisões estratégicas. Por este motivo, se tornam cada vez mais necessários serviços que mantenham alta disponibilidade de acesso a essas informações.

O principal objetivo de um banco de dados é armazenar informações de forma segura e íntegra, mantendo disponível seu acesso quando necessário (KORT; SILBERSCHATZ; SUDARSHAN, 2006).

De acordo com Casanova e Moura (1999), desde 1970 a arquitetura centralizada de banco de dados é comumente utilizada nos ambientes corporativos. Porém, esse panorama está sendo modificado. Com o crescimento das organizações, cada dia mais é necessário a descentralização da informação, mantendo sua disponibilidade em ambientes distintos geograficamente. Como solução para atender a essa necessidade de integração e disponibilidade, vêm-se adotando a utilização de bancos de dados distribuídos.

Banco de dados distribuído é uma tecnologia baseada na junção de duas tecnologias: banco de dados, que possui o propósito de manter as informações de forma centralizada, e a rede de comunicação que constitui uma nova forma de acesso à informação (NUMATA, 2012).

Na estrutura abordada no estudo de caso deste trabalho, é utilizado o conceito de

[1] IDV (Instituto de Desenvolvimento do Varejo) - Instituto criado em 2004 com o propósito de desenvolver o crescimento sustentável do setor de varejo.

[2] PIB (Produto interno bruto) – É a soma de todas as riquezas produzidas numa determinada região durante um período determinado (IBGE, 2016).

[3] IBGE - Instituto Brasileiro de Geografia e Estatística.

[4] Em todo o trabalho é utilizado o termo "software", ao invés de "programa de computador". Foi feita essa opção pelo fato desse termo ter seu uso generalizado em periódicos, em pesquisas científicas e outros tipos de publicações, inclusive de cunho jurídico (SALEH, 2004).

bancos de dados distribuídos, que utiliza o mecanismo de replicação nativa do MySQL para manter os dados sincronizados entre as bases de dados. O ambiente de replicação utilizado no estudo de caso é o *Master*/*Slave* em que o banco de dados Master é responsável por receber a gravação das informações e as consultas são realizadas banco de dados *Slave*. Essa arquitetura possibilita diminuir o tráfego da rede, o tempo referente à busca de informações e a carga do servidor central. Entretanto, o crescimento do setor impacta diretamente no volume e tráfego de informações, e manter aplicações que efetivam as transações de forma *online* pode afetar o desempenho, a integridade e a disponibilidade da aplicação.

Enterprise Integration Partners EIPs[5] se refere a um conjunto de padrões de integrações corporativa que tem como objetivo simplificar o processo de comunicação entre sistemas.

Esse trabalho apresenta o desenvolvimento de um mecanismo que, através dos padrões EIPs proporciona a integração de informações entre bancos de dados MySQL[6], esse recurso consiste em sincronizar algumas informações entre os bancos de dados que participam da integração.

O mecanismo de integração busca diminuir a dependência de uma base de dados central. Ele possui a estratégia de integrar apenas o que é pertinente àquele ambiente, aplicando técnicas de fragmentação de dados, que consistem em manter a informação próxima do local de sua utilização. Mesmo com um cenário distribuído, algumas informações têm a necessidade de estar em todas as bases de dados, por possuírem um valor significativo para a regra de negócio do estudo de caso.

O trabalho atual consiste em demonstrar as técnicas de integração de sistemas como uma alternativa para um processo de sincronização de informações.

As demais seções desse trabalho estão organizadas da seguinte maneira: na Seção 2 são apresentados os trabalhos relacionados; na Seção 3 a descrição do mecanismo de integração de dados e seu funcionamento; a Seção 4 é composta pela aplicação do mecanismo no estudo de caso; na Seção 5 é apresentada a análise dos resultados e na Seção 6 são apresentadas as conclusões e considerações finais do trabalho.

2. TRABALHOS RELACIONADOS

Bortolini (2004) desenvolveu um protótipo baseado em um estudo de caso,

[5] EIPs (*Enterprise Integration Partners*) - conjunto de padrões utilizados no processo de integração de sistemas corporativos (HOHPE; WOOLF, 2003).
[6] MySQL - servidor e gerenciador de banco dados relacional (MILANI, 2007).

em que existia a necessidade de manter as informações sincronizadas entre as filiais de uma determinada empresa. Com a finalidade de manter no banco de dados das filiais apenas as informações pertinentes a sua utilização. O auto plicou as técnicas de fragmentação de dados horizontal somente nas informações relevantes para determinado local, nas demais situações, as informações permaneceram íntegras.

Para manter o controle da distribuição de dados, segundo o autor, o protótipo utiliza a arquitetura *Master*/*Slave*, que por sua vez proporciona um controle maior sobre a informação, podendo ser realizado um processamento da informação antes de ser enviada para os bancos de dados locais, por meio de um mecanismo de replicação assíncrona. Nesse cenário, o banco de dados é único na rede, sendo responsável por receber e armazenar todas as transações das lojas. A cada nova transação, o próprio SGBD[7] gera *logs*[8] que poderão ser utilizados no processo de replicação.

Nessa estrutura, a gravação de dados só é permitida no banco de dados "master_db". Os servidores "slave_sb1", "slave_db2" e "slave_db3" somente recebem as atualizações e as mantem disponíveis para leitura.

Esse processo de atualização entre o servidor *master* e os servidores *slaves* é realizado através de um serviço nativo do próprio MySQL. Nesse cenário, é utilizada a replicação assíncrona através da comunicação direta entre os servidores (MySQL, 2016).

Para realizar a replicação na arquitetura *Master*/*Slave* o autor não utilizou o mecanismo nativo oferecido pelo SGBD, optando por utilizar uma ferramenta externa de suporte a replicação, o IBReplicator, que possui os seguintes agentes:

- Gerente de Replicação: ferramenta que pode ser instalada em qualquer computador de uma rede, ela abstrai uma camada de acesso ao servidor sendo responsável por receber todas as requisições ao banco de dados;
- Servidor de replicação: ferramenta escrita em linguagem C que usa diretamente as *APIs*[9] do próprio SGBD, dispensando desta forma o uso de *middleware*[10], como JDBC[11] ou BDE[12]. Esse mecanismo somente pode existir na máquina onde está instalado o banco de dados, e é

[7] SGBD (Sistema Gerenciador de Banco de Dados) - O sistema de gerenciamento de banco de dados é o software que trata de todo o acesso ao banco de dados (DATE, 2003).

[8] *Logs* - aqui se referindo a *log* binário, um arquivo responsável por armazenar o histórico de instruções processadas pelo banco de dados (MySQL, 2016).

[9] API (*Application Programming Interface*) - Conjunto de rotinas e padrões que fornecem uma grande facilidade ao programador (MENDES, 2009).

[10] Middleware - termo normalmente utilizado para um código de software que atua como mediador entre dois programas de computador.

[11] JDBC (*Java Database Connectivity*) - uma API java que possibilita a integração com banco de dados.

[12] BDE (*Borland DataBase Engine*) - Software responsável por permitir a comunicação com banco de dados baseado em Windows.

responsável por manter as bases de dados sincronizadas;

Freitas (2003) demonstrou a proposta de uma arquitetura de banco de dados distribuída baseada em réplicas do servidor principal. No trabalho, não foram aplicadas regras de fragmentação de dados. O autor enfatiza que apesar de aumentar a duplicidade das informações, a ausência da fragmentação possui um fator positivo, quando a arquitetura do sistema exige uma alta taxa de leitura de dados.

O mecanismo adotado pelo autor para manter as bases de dados sincronizadas baseia-se no modelo de comunicação RMI (*Remote Method Invocation*), mecanismo de compartilhamento de métodos entre aplicações Java[13] utilizando máquina virtual Java (JVM)[14]. De acordo com o autor, RMI disponibiliza uma família de objetos colaborativos que podem ser acessados por meio de um protocolo padrão de rede. O método permite a comunicação entre máquinas virtuais Java executadas em computadores distintos. O projeto utiliza esse recurso para gerenciar e coordenar as atualizações e consultas no ambiente distribuído, seguindo os seguintes conceitos:

- Nó: cópias idênticas de banco de dados armazenados em diferentes locais;
- Cliente: camada de apresentação para utilização do sistema, onde o usuário realiza consulta e atualiza as informações;
- Servidor: camada responsável por gerenciar e disponibilizar as informações armazenadas no banco de dados.

Freitas (2003) destaca a dificuldade de garantir a atomicidade em um ambiente distribuído, como forma de contornar este problema. O autor utiliza um mecanismo de escrita de *logs*, que consiste na criação de um arquivo com todas as alterações realizadas, e em caso de falha na atualização de um nó, devido estar indisponível no momento da replicação, o arquivo de *log* pode ser utilizado como forma de sincronizar os dados.

3. DESCRIÇÃO DO MECANISMO DE INTEGRAÇÃO DE DADOS

O mecanismo de integração proposto neste trabalho possui a responsabilidade de manter os dados pertinentes de uma integração sincronizada entre bases de dados MySQL.

O banco de dados MySQL é um sistema de banco de dados relacional que surgiu na década de 90. No inicio de sua

[13] Java, linguagem de programação orientada a objeto, multithread, interpretada neutra de arquitetura (MENDES, 2009).
[14] JVM (*Java Virtual Machine*) representa a base da plataforma Java, sendo responsável por interpretar e executar os programas Java (MENDES, 2009).

utilização o banco de dados MySQL era recomendado para aplicações de pequeno a médio porte, podendo armazenar em sua estrutura aproximadamente até 100 MBytes por tabela e também capaz de armazenar em suas tabelas algo em torno de 100 milhões de registros. A versão mais recente 5.7 do MySQL utiliza o padrão SQL-92 ODBC levels 0-3.51, o principal objetivo do SGBD é utilizar o padrão SQL-99, porém, nem todos os códigos podem ser alterados para o novo padrão sem que seja comprometida a velocidade e confiança do banco de dados (MILANI, 2007).

O processo de integração será realizado utilizando a arquitetura apresentada na seção 2, de bancos de dados *Master*/*Slave*. O servidor *Master* não depende de comunicação com outros servidores para o seu funcionamento e possui o mecanismo de escrita de *log* binário que pode ser utilizado no processo de replicação de dados. Já o servidor *Slave*, precisa manter uma comunicação direta com o servidor *Master* e através da interpretação dos arquivos de *log* binário, mantém os registros atualizados.

Porém, nesse caso, todos os servidores serão instalados como *Master*, ou seja, cada um será considerado como servidor principal, sem a necessidade de um processo de replicação para manter os dados sincronizados. O mecanismo será responsável por receber as consultas e atualizações dos usuários conectados.

Como este trabalho foi baseado em estudo de caso de uma empresa de desenvolvimento de software que possui um software legado, buscou-se evitar sugerir alterações em seu produto referente à criação de chaves primárias. Desta forma, como solução para o cenário de gerar chaves duplicadas entre as bases de dados, foi utilizado um recurso que surgiu a partir do MySQL 5, que se refere a um controle de geração de chaves a partir das propriedades *auto_increment_increment* e *auto_increment_offset* que são responsáveis por gerar um par de chaves.

A linguagem de programação utilizada no desenvolvimento desse projeto foi Java, por atender às necessidades de independência de sistema operacional e independência de *hardware*, pois existe uma JVM (*Java Virtual Machine*) para cada sistema operacional. Além disso, a linguagem implementa o conceito de *multithread*, permitindo o paralelismo entre os processos realizados pela aplicação (MENDES, 2009).

Para realizar o processo de integração, foi desenvolvido um mecanismo que se baseia no padrão de EIPs de troca de mensagens entre os sistemas, utilizando como base o *framework*[15] Apache Camel que,

[15] *Framework* - conjunto de bibliotecas que proporciona e simplifica a execução de determinadas tarefas.

por sua vez, possui a responsabilidade de gerenciar toda a arquitetura de envio e recebimento da informação, encapsulando toda a complexidade da troca de mensagens.

A integração é um mecanismo que visa proporcionar ao usuário uma visão unificada do modelo de negócio, mesmo onde as plataformas são distintas e baseadas em tecnologias diferentes ou geograficamente distantes. Por isso, não existe um modelo de integração a ser seguido, deve-se realizar a escolha mais apropriada para cada integração, baseada, segundo Hohpe e Woolf (2003), nos seguintes critérios:

- Formato de dados: o processo de integração exige que as aplicações utilizem o mesmo formato de dados. Em alguns casos, alteração do formato de dados das aplicações é inviável, sendo necessário aplicar tradutores no processo de integração;
- Seleção da tecnologia: integrações podem exigir diferentes licenças se *software* e *hardware*, podendo aumentar o custo da integração e tornar o projeto dependente de fornecedores, mas também pode aumentar a curva de aprendizado da equipe desenvolvedora;
- Exposição da funcionalidade: integrações permitem o compartilhamento de dados e de funcionalidades, desta forma, uma aplicação pode utilizar métodos de outra aplicação, da mesma maneira como invoca um método local;
- Tempo para atualização: quando a integração refere-se a compartilhamento de dados, o tempo de atualização deve ser o menor possível, para não resultar em problemas referentes a sincronismo. O ideal é que o receptor seja informado o mais rápido possível quando os dados compartilhados estiverem prontos para consumo;
- Confiabilidade: conexões remotas não são apenas lentas, mas também são muito menos confiáveis do que uma conexão local. A execução de chamadas remotas está mais suscetível a erros, devido à indisponibilidade das aplicações remotas ou da rede de comunicação;
- Comunicação assíncrona: permite que o aplicativo de origem continue seu trabalho realizando outra tarefa, de forma confiante, pois a aplicação destino receberá a informação;
- Processamento assíncrono: traz vantagens como a escalabilidade, mas é um processo que possui *design*, desenvolvimento e depuração mais complexos;
- Acoplamento entre as aplicações: as aplicações devem minimizar suas

dependências, de modo que cada aplicação possa evoluir sem causar problemas para as demais. As integrações devem ser específicas o suficiente para cumprir seu papel e, genéricas o suficientes para garantir que mudanças não façam as aplicações dependentes pararem;

- Intrusividade: as integrações devem causar o mínimo de impacto em código existente e exigir pouca codificação.

De acordo com Hohpe e Woolf (2003), não existe uma abordagem de integração que contemple todos os critérios ao mesmo tempo, por isso, deve-se escolher a que mais se adequa ao cenário. As abordagens de integração podem ser resumidas em quatro categorias:

- *File Transfer* ou Transferência de Arquivos: uma aplicação produz arquivos de dados para outra aplicação consumir, onde a primeira exporta um arquivo a ser importado pela segunda;
- *Remote Procedure Invocation* ou Chamada Remota de Procedimentos: as aplicações mantêm alguns de seus métodos disponíveis para que as outras aplicações possam invocar remotamente;
- *Shared Database* ou Integração entre Bases de Dados: as aplicações gravam em um banco de dados de comum acesso as informações que desejam compartilhar;
- *Messaging* ou Troca de Mensagens: as aplicações se conectam a um sistema comum de mensageria, compartilhando dados e invocando operações através do uso de mensagens.

O processo de integração deste trabalho é baseado em um sistema de mensageria, que de acordo com Hohpe e Woolf (2003) é a melhor abordagem para integrações empresariais. Dessa forma, como requisito essencial, o administrador inicialmente deve definir uma fila JMS[16] para receber as alterações dos bancos de dados e outra para cada servidor receber as alterações realizadas nos outros bancos de dados.

A comunicação JMS suporta dois tipos de comunicação que são tratados como filas:

- *Queue*: comunicação ponto a ponto, onde um produz as mensagens e as coloca em uma fila JMS a ser processada por um consumidor;
- *Topics*: modelo onde um produtor é responsável por gerar as mensagens e disponibilizá-las para um ou mais consumidores.

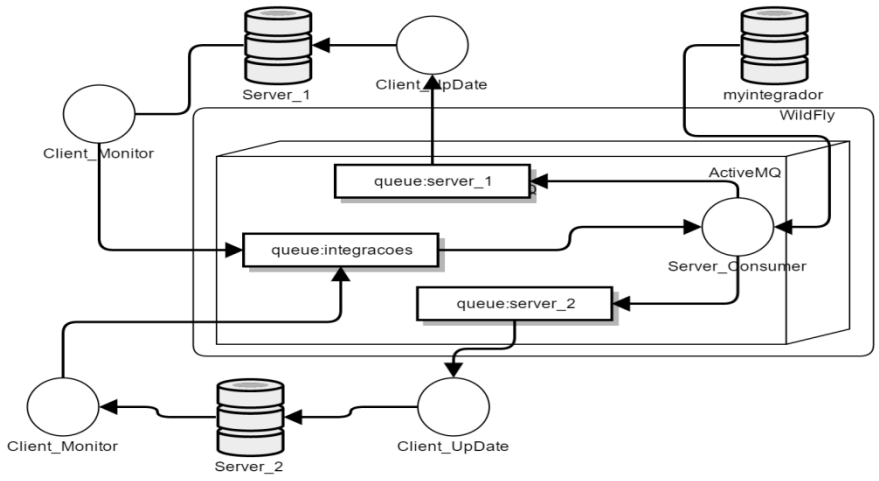
[16] JMS (*Java Message Service*) - API da linguagem Java que proporciona a comunicação entre duas ou mais aplicações através de troca de mensagens.

Para que a comunicação JMS funcione corretamente é necessário que exista um provedor responsável por gerenciar as sessões e filas, dentre as opções livres disponíveis, este projeto utiliza o ActiveMQ.

Na Figura 1, é demonstrado o cenário de integração proposto neste trabalho, onde existem os seguintes componentes:

- *Server_1* e *Server_2*: banco de dados que participam da integração e devem ter suas informações sincronizadas;
- *Client_Monitor*: serviço conectado diretamente ao bancos de dados *Server_1* e *Server_2* responsável por processar a integração cadastrada e gerar um XML de cada entidade pertencente à integração;
- *Client_UpDate*: serviço responsável por monitorar uma fila de integração específica, e executar os *scripts* recebidos na base de dados configurada;
- *Server_Consumer*: serviço disponível somente no servidor de aplicação, responsável por rotear as mensagens enviadas pelo *Client_Monitor* para a fila de *queue:integrações* para as filas cadastradas para os outros bancos de dados);
- *MyIntegrador*: banco de dados que alimenta as configurações necessárias para que a integração seja realizada corretamente;
- *WildFly*: servidor de aplicação JEE (*Java Enterprise Edition*), que permite a execução de aplicação que siga a especificação JEE.

Figura 1. Componentes para integração.



O projeto conta com uma etapa de configuração, que deve ser armazenada na tabela *myintegrador.* As informações referentes à integração estão armazenadas no banco de dados nas tabelas *integracoes* e *tabelas.* O registro armazenado na tabela *integracoes* com o campo "sequencia" com valor zero é considerado como *header* da integração, ou seja, registro principal e através dele que as demais informações devem ser carregadas. A tabela *database* é responsável por armazenar os registros referentes aos bancos de dados que participam da integração e é utilizada pelo serviço *Server_Consumer* para identificar as filas que cada banco de dados monitora.

Como requisito básico das integrações, o trabalho apresenta uma *flag* de integração. A *flag* de integração baseia-se em um atributo a ser inserido na entidade principal da integração e possui a responsabilidade de ser o gatilho para que o serviço responsável por monitorar as integrações identifique que o registro foi alterado, e envie para os outros bancos de dados.

A *flag* de integração possui os seguintes valores:

- 0 (zero) – indica que o registro não teve alteração;
- 1 (um) – indica que o registro foi incluído;
- 2 (dois) – indica que o registro foi alterado.

A chave estrangeira *foreign key*[17] possui a responsabilidade de manter a ligação entre as tabelas relacionadas na integração, e essas informações são utilizadas no momento de carregar cada entidade.

Para não ser necessário modificar os aplicativos que alimentam as fontes de dados, será utilizado o mecanismo de *trigger*[18] do próprio SGBD como forma de monitorar as tabelas pertencentes à integração. Pode-se observar nas Figuras 2 e 3 que a *trigger* possui a responsabilidade de monitorar o banco de dados e aplicar o status no campo *status_integacao* de acordo com a operação realizada, essa operação é realizada nas tabelas que pertencem à integração.

Figura 2. *Trigger* que controla os eventos de inclusão de registros na tabela de clientes.

```
BEGIN
  IF (NEW.STATUS_INTEGRACAO = 0)  THEN
      SET NEW.STATUS_INTEGRACAO = 1;
    END IF;

END
```

[17] *Foreign key* ou chave estrangeira tem por finalidade estabelecer a ligação entre duas entidades do banco de dados. MySQL (2016).

[18] *Trigger*, gatilho associando a eventos de uma tabela MySQL (2016).

Figura 3. *Trigger* que controla os eventos de atualização de registros na tabela de clientes.

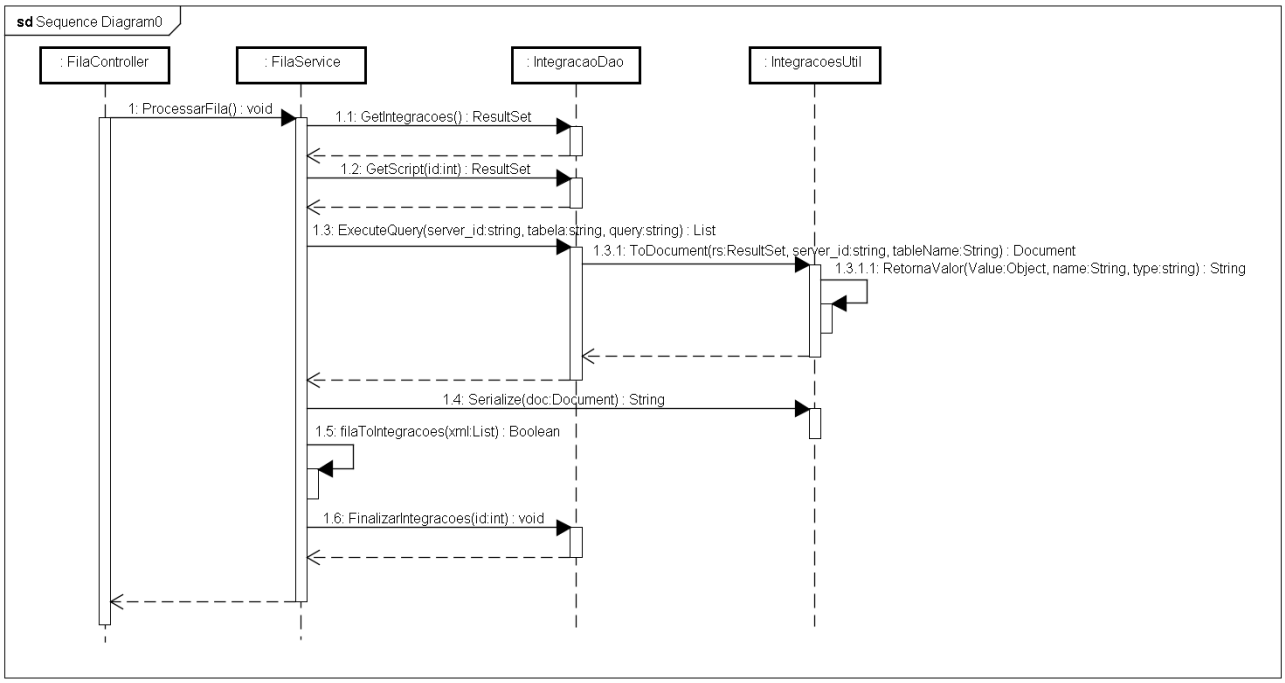
```
BEGIN
  DECLARE ATUALIZOU BOOLEAN DEFAULT FALSE;
    IF (NEW.STATUS_INTEGRACAO = 0) AND (OLD.STATUS_INTEGRACAO = 0) THEN
        SET NEW.STATUS_INTEGRACAO = 1;
    SET ATUALIZOU = TRUE;
  END IF;
    IF NOT ATUALIZOU THEN
          IF (OLD.STATUS_INTEGRACAO = 1) AND(NEW.STATUS_INTEGRACAO <> 0)  THEN
              SET NEW.STATUS_INTEGRACAO = 2;
                  SET ATUALIZOU = TRUE;
        END IF;
  END IF;

 IF NOT ATUALIZOU THEN
      IF (OLD.STATUS_INTEGRACAO = 2) AND(NEW.STATUS_INTEGRACAO = 2)  THEN
              SET NEW.STATUS_INTEGRACAO = 2;
    END IF;
 END IF;

END
```

Na Figura 4, pode-se observar as etapas executadas pelo serviço *Client_Monitor*, que tem como objetivo de verificar os registros alterados e enviá-los para a fila de integrações (*queue:integracoes*).

A partir do diagrama da Figura 4, destaca-se os principais métodos:

- *GetScript*: responsável por carregar os dados da integração em processamento. Para obter as informações corretamente de cada entidade relacionada na integração, utiliza a tabela do MySQL *information_schema*, que armazena as informações relacionadas a estrutura da tabela, possibilitando carregar as tabelas e seus relacionamentos;

Figura 4. Diagrama de sequência do projeto *Client_Monitor*.



- *ToDocument*: responsável por transformar o resultado de uma consulta SQL em um arquivo XML que será enviado para a fila *queue:integracoes.*

- *FilaToIntegracoes*: recebe lista de XML que contém a informação referente a integração que está sendo processada para a fila de integrações *queue:integracoes.* O método utiliza a

API Apache Camel através do componente *ProducerTemplate*, dessa forma é possível abstrair todas as complexidades e implementações que um mecanismo de mensageria exige.

O serviço responsável por realizar o roteamento da fila *queue:integracoes* é o *Server_Consumer*, que envia a mensagem recebida para a fila de integração de cada banco de dados, exceto para o banco de dados que enviou a mensagem.

A implementação realizada é considerada simples, pois utiliza o *framework* Spring para simplificar o processo de monitoramento da fila *queue:integracoes*.

4. EXPERIMENTO

Nesta seção é apresentado um experimento em laboratório baseado em um estudo de caso de uma empresa desenvolvedora de software que atua no seguimento farmacêutico desde 1994. Como se trata de um setor composto por um grande número de franquias e filiais, a empresa disponibiliza para seus clientes um recurso multi-loja, que consiste em manter as informações entre as lojas de forma sincronizada, através de uma arquitetura de banco de dados centralizada. Este cenário utiliza o recurso de *Master* e *Slave*, anteriormente abordados na segunda seção desse trabalho.

O mecanismo foi adotado pela empresa no ano de 2008 para atender clientes em potencial no mercado, elevando o produto desenvolvido a trabalhar *on-line*. Porém, com o crescimento do seguimento varejista, o aumento no volume e tráfego de informações, manter as aplicações gravando e efetivando suas transações de forma *on-line* e síncrona pode afetar os seguintes requisitos:

- Desempenho: como todas as transações devem ser efetivadas no servidor central, ele passa a ser sobrecarregado pelo número excessivo de transações simultâneas;
- Integridade: todas as informações devem ser efetivadas no servidor central, tornando-o vulnerável a problemas de perda e oscilação de comunicação;
- Disponibilidade: como todas as transações são efetivadas em único banco de dados, caso o servidor fique inacessível, os aplicativos passam a trabalhar somente em modo consulta em algumas operações.

Como forma de contornar os problemas citados e ainda garantir a integração da informação entre as lojas, foi proposto neste trabalho uma nova arquitetura, que consiste em aplicar o

modelo de banco de dados distribuídos em conjunto com as técnicas de fragmentação de dados, em que cada banco de dados deve armazenar as informações de cada loja, tornando o acesso e a atualização das informações mais rápidas.

Para realizar a sincronização de dados entre duas lojas (Loja 1 e Loja 2) foi utilizado o padrão de integração entre sistemas *messaging,* já mencionado na Sessão 3, deste trabalho.

Para demonstrar as funcionalidades do mecanismo de integração proposto neste trabalho, foram mapeadas algumas integrações relativas às regras de negócio de setor farmacêutico, tais como: clientes, produtos, estoque, permissões, recebimentos e contas pendentes de clientes. Com a integração das informações relacionadas, será possível atender os seguintes cenários:

- O cliente realiza compras em todas as lojas utilizando o mesmo cadastro;
- O cliente realiza o pagamento total de suas contas pendentes em uma loja, mesmo existindo compras em lojas distintas;
- O administrador poderá controlar as permissões de usuário de forma unificada para todas as lojas;
- Cadastro unificado de produtos que proporciona melhor controle de preços e avaliação do estoque da loja.

Para demonstrar as funcionalidades do trabalho, primeiramente faz-se necessário adequar o cenário, com as seguintes etapas:

- Cópia dos dados: consiste em realizar um backup das informações do banco de dados central, para ser aplicado no servidor das filiais;
- Instalação do banco de dados: como o mecanismo de replicação do SGBD MySQL. Não foi necessário reinstalar o serviço MySQL nas filiais utilizando a versão 5.7.13, modificando a estrutura do servidor *Slave* para *Master*. A instalação do banco de dados *Master* foi ajustada para receber as configurações de controle de chaves, que representa o banco de dados da Loja 1 e Loja 2;
- Análise da integração: após a instalação do servidor MySQL foi necessário verificar se os relacionamentos das tabelas pertencentes às integrações (clientes, produtos, estoque, permissões, recebimentos e contas pendentes de clientes) estavam informados de forma correta. Na integração proposta, foram encontrados os relacionamentos exemplificados nas Figuras 5, 6, 7 e 8.

Figura 5. Relacionamento para integração de clientes.

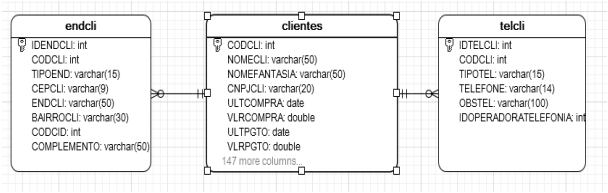


Figura 6. Relacionamento para integração de contas pendentes.

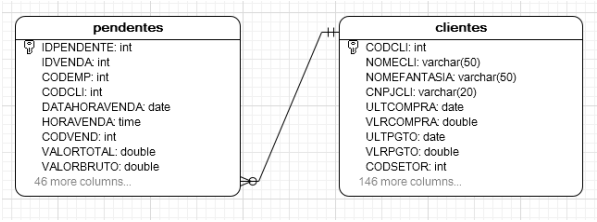


Figura 7. Relacionamento para integração do estoque.

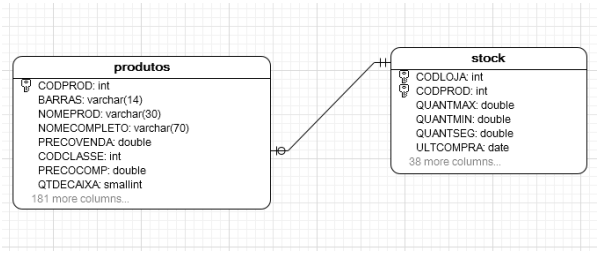
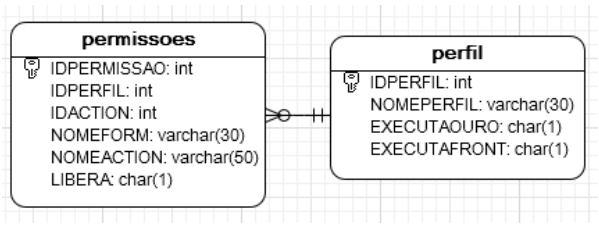


Figura 8. Relacionamento para integração das permissões.



Para cada integração descrita nas Figuras 5, 6, 7 e 8 faz-se necessário incluir o campo "STATUS_INTEGRACAO" nas tabelas "CLIENTES", "PRODUTOS", "STOCK", "PENDENTES", "PERMISSOES" e "PERFIL", e criar as *triggers* mostradas nas Figuras 2 e 3, que são responsáveis por monitorar as integrações.

Após a instalação do servidor de banco de dados na loja 1 e na loja 2 e realizadas todas as adequações, neste ponto a estrutura da loja já está apta para receber os aplicativos responsáveis por monitorar as integrações *Client_Monitor* e *Client_UpDate*.

Como forma de coletar indicadores, foram realizadas as seguintes operações no ambiente de replicação e no de integração:

- Cadastro de um novo cliente;
- Atualização dos dados do cliente;
- Realização de venda para o cliente na modalidade a prazo;
- Criação de um novo perfil de usuário;
- Atualizadas informações referentes a permissões;
- Cadastrado um novo produto;
- Atualização dos dados cadastrais do produto;
- Realização da venda com produto.

Como forma de manter a similaridade da estrutura, ambos os experimentos foram realizados em um ambiente virtual com configurações idênticas.

No processo de cadastro de novas informações pode ser observado o funcionamento configurado em cada servidor para o controle de chaves. Nas Figuras 9 e 10, pode-se observar o resultado do auto

incremento executado por cada banco de dados, em que ambos os bancos de dados iniciaram sua sequência na chave 18251.

Figura 9. Tabela Clientes da Loja 1.

CODCLI	NOMECLI	NOMEFANTASIA	CNPJCLI	ULTCOMPRA
18251	TESTE CADASTROS		04.825.836/0001-86	2016-07-05
18253	IRIS FURTADO	IRIS PÉGO	26434513500	(Null)
18255	ILDA FURTADO	ILDA GENEROSO	80921218346	(Null)

Figura 10. Tabela Clientes da Loja 2.

CODCLI	NOMECLI	NOMEFANTASIA	CNPJCLI	ULTCOMPRA
18251	TESTES DE CADASTRO		04.825.836/0001-8	2016-07-05
18252	DAVI FURTADO	DAVI	21562220535	(Null)
18254	VIVIANE FURTADO	VIVI	26434513500	(Null)

Ao modificar alguma informação no banco de dados da Loja 1 e Loja 2 onde a informação modificada pertença a uma integração cadastrada, o aplicativo *Client_Monitor* envia os dados para uma fila JMS *queue:integracoes*.

A fila de integração da loja é monitorada pelo aplicativo *Client_UpDate,* que é responsável por receber as novas atualizações e efetiva-las no banco de dados conectado.

O *Server_Consumer* monitora a fila *queue:integrações,* onde são recebidas informações do banco de dados da Loja 1 e enviadas para a fila de *integrações_2* monitorada pela Loja 2, aguardando ser processada. O *Client_UpDate* da Loja 2 recupera as informações pendentes da fila JMS *queue:integrações_2* que devem ser inseridas no banco de dados, o resultado da integração de clientes pode ser observado na figura 11.

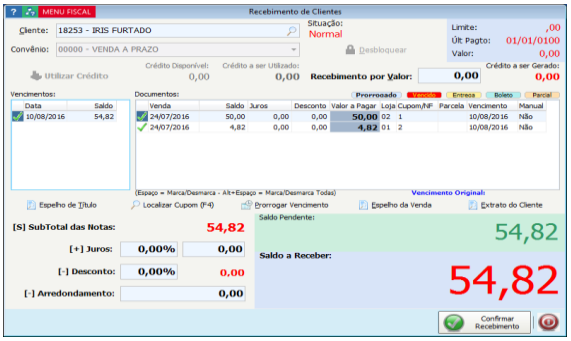
Figura 11. Resultado integrações de clientes tabela Clientes Loja 2.

CODCLI	NOMECLI	NOMEFANTASIA	CNPJCLI	ULTCOMPRA
18251	TESTES DE CADASTRO		04.825.836/0001-8	2016-07-05
18252	DAVI FURTADO	DAVI	21562220535	(Null)
18253	IRIS FURTADO	IRIS	07763492104	(Null)
18254	VIVIANE FURTADO	VIVI	26434513500	(Null)
18255	ILDA FURTADO	ILDA GENEROSO	80921218346	(Null)

Com a integração dos clientes, ambas as lojas podem efetuar vendas para os clientes e suas compras aparecem de forma unificada.

Para comprovar o resultado da integração de contas parceladas, foram realizadas duas vendas na Loja 1 e na Loja 2, respectivamente, para um cliente cadastrado inicialmente na Loja 1. Foi possível observar o resultado da integração no momento de realizar o recebimento das compras a prazo do cliente, em que as duas vendas apareceram para o usuário realizar a quitação do débito conforme figura 12.

Figura 12 – Recebimento de Clientes



As integrações de produtos e permissões seguiram os fluxos abordados nas integrações já mencionadas e proporcionaram os mesmos resultados.

5. RESULTADOS

Foi observado que a base de dados utilizada pelo estudo de caso não implementa o conceito de chave estrangeira *foreign key*, desta forma a primeira modificação a ser realizada foi adequar as tabelas pertencentes a integração. Após esta etapa ainda foi necessário realizar a criação das *triggers* nas tabelas envolvidas na integração. Esse processo foi realizado de forma manual através da interface de gerenciamento do próprio SGBD, mas poderia ser melhorado na continuação do mecanismo como mais uma funcionalidade do gerenciador de integração, gerando os scripts da *trigger* de forma automatizada sendo executado pelo aplicativo *Client_UpDate* na base de dados configurada.

Outro ponto que teve que ser adequado para que a integração funcionasse de forma correta, foi e elaboração de um controle de envio de informações para a fila de integração, garantindo que os dados das tabelas auxiliares fossem gravados no banco de dados após a existência do registro principal, pois caso tal requisito não fosse resolvido no momento de gravar as informações auxiliares, o banco de dados apresentaria um erro de referência devido às validações aplicadas nas chaves *foreign key*.

Como forma de resolver o problema de integridade do banco de dados, foi utilizado o recurso de *Custom-Priority* do componente do Apache Camel. Tal recurso consiste em informar que os registros principais tinham maior prioridade e deviam ser integrados e processados primeiramente na fila JMS.

Foi observado no experimento que o consumo de recursos da máquina onde foram instalados os serviços *Client_Monitor* e C*lient_Update*, em um estado ocioso, uma alteração no consumo de memória RAM que passou de 44% para 64% e uma variação mínima na escrita de disco, como pode ser observado na Figura 13 e 14 que representam a evolução do gerenciador de tarefas do computador utilizado no experimento.

Figura 13. Gerenciador de Tarefas sem a execução dos serviços de integração.

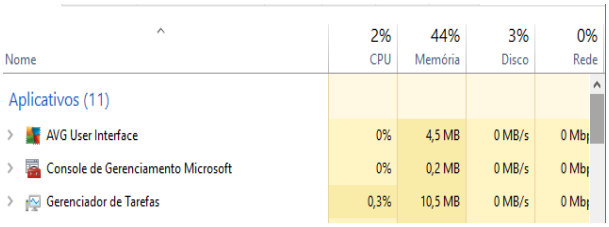
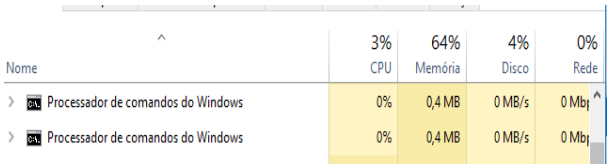
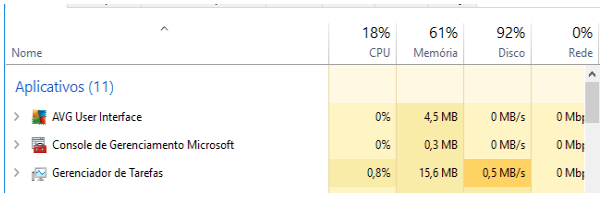


Figura 14. Gerenciador de Tarefas com a execução dos serviços de integração.



Com a integração sendo executada, foi observado um aumento da escrita em disco que passou de 4% para 92% conforme pode ser observado nas Figuras 15.

Figura 15. Gerenciador de Tarefas integração sendo executada.



Com um grande volume de integrações esse seria um panorama constante, prejudicando a utilização do computador em outros processos.

No ambiente de replicação *Master*/*Slave* o consumo de recurso do servidor *Slave* não apresentou modificações devido o serviço de replicação ser nativo do próprio SGBD. O mecanismo de replicação não afetou o desempenho do computador, além do MySQL oferecer vários parâmetros personalizados de configuração para garantir o melhor resultado para o ambiente.

Os processos de gravação de dados mantiveram seu desempenho inalterado, porém foi possível observar que a oscilação do *link* de internet gera um travamento da aplicação no processo de cadastro de dados. No mecanismo de integração tal oscilação não causa nenhum tipo de anomalia, tendo em vista que o servidor de gravação está em uma rede local.

No processo de replicação, em caso de falha no *link* de internet, o serviço de cadastro de novas informações fica comprometido, tornando o sistema *off-line*. Neste cenário o sistema somente realiza vendas que serão efetivadas quando o cenário for normalizado e consequentemente as informações param de ser sincronizadas com outros bancos de dados.

No cenário de integração a falha de *link* de internet não compromete as operações da aplicação, porém o processo de envio e recebimentos de novas informações fica aguardando o cenário ser normalizado para retomada da integração.

6. CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi analisar o recurso de integração de dados como substituto para o processo de replicação de informações utilizado em um ambiente corporativo. Este trabalho demonstrou como é possível aplicar as técnicas de EIPs em um

ambiente corporativo, eliminando a dependência de um banco de dados central sem exigir modificações no software utilizado. Para aplicar o modelo proposto, foram necessárias algumas alterações estruturais na instalação do banco de dados das filiais, um processo de adequação da fonte de dados, criação das f*oreign key* e das *trigger*.

Com a utilização do mecanismo proposto, foi possível manter a compatibilidade entre as bases de dados das lojas, e as regras de negócio do segmento à medida que tal alteração não transparecesse para o usuário.

O mecanismo de integração desenvolvido neste trabalho foi baseado no estudo de caso de um software já em produção no segmento farmacêutico, e para uma elaboração eficiente, foi necessário entender e preservar as características do software para que não gerasse impacto em sua utilização.

A maior diferença entre os dois ambientes analisados neste trabalho se refere ao consumo de recursos extras que a integração de dados exige para efetuar suas tarefas. Desta forma pode-se concluir que o processo de integração pode ser utilizado como opção para sincronização de dados entre sistemas, porém para cada cenário deve ser realizado um estudo para preservar as funcionalidades do ambiente, tornando complexo um processo genérico que pudesse ser utilizado em ambientes diferentes.

REFERÊNCIAS

BORTOLINI, C. A. Um Protótipo de banco de Dados Distribuídos (Caso Expresso São Miguel). 2004. 101f. Monografia (Ciência da Computação) – Universidade Comunitária Regional de Chapecó, Chapecó – SC, 2004.

CASANOVA, M. A.; MOURA, A. V. Princípios de Sistema de Gerência de Banco de Dados Distribuídos. Rio de Janeiro: Campus, 1999.

DATE, C. J. Introdução a sistemas de bancos de dados. 8. ed. Rio de Janeiro: Elsevier, 2003.

FREITAS, A. L. C. Proposta de um sistema de banco de dados distribuído e replicado utilizando serviço RMI-Java. 2003.

IDV. A década de Ouro do Varejo - 10 anos do IDV. 2014. Disponível em: <http://www.idv.org.br/releases/a-decada-de-ouro-do-varejo-10-anos-do-idv/>. Acesso em: 04 jul. 2016.

IBGE. As divisões do produto interno bruto brasileiro. Instituto Brasileiro de Geografia e Estatística. 2015. Disponível em: <http://teen.ibge.gov.br/en/biblioteca/livros-on-line/272-teen/noticias/2817-as-divisoes-do-produto-interno-bruto-brasileiro.html>. Acesso em: 04 jul. 2016.

HOHPE, G.; WOOLF, B. Enterprise Integration patterns designing, building, and deploying messaging solutions. 1. ed. Canada: Addison-Wesley Professional, 2003.

MENDES, D. R. Programação Java com Ênfase em orientação a objetos. 1. ed. São Paulo: Novatec, 2009.

MILANI, A. MySQL Guia do Programador. 1. ed. São Paulo: Novatec, 2007.

MySQL. MySQL 5.5 Reference Manual, Oracle Corporation. 2016. Disponível em: <http://dev.mysql.com/doc/refman/5.5/en/>. Acesso em: 02 jul. 2016.

KORT, H. F.; SILBERSCHATZ, A.; SUDARSHAN S. Sistema de banco de dados. 5. ed. Rio de Janeiro: Campus, 2006.

NUMATA, C. A. Banco de dados distribuídos. 2012. 41f. Monografia (Processamento de Dados) - Faculdade de Tecnologia de São Paulo, São Paulo – SP, 2012. Disponível em: <http://www.fatecsp.br/dti/tcc/tcc00063.pdf>. Acesso em: 10 jan. 2016.

SALEH, A. M. Adoção de tecnologia: um estudo sobre o uso de software livre nas empresas. 2004. Dissertação (Mestrado) – Faculdade de Economia, Administração e Contabilidade. São Paulo: Universidade de São Paulo, 2004.