UNIVERSIDADE FEEVALE

MOISÉS FELIPE LEHNEN

PROPOSTA DE REPLICAÇÃO DE DADOS EM AMBIENTE HETEROGÊNEO

Novo Hamburgo

2011

MOISÉS FELIPE LEHNEN

PROPOSTA DE REPLICAÇÃO DE DADOS EM AMBIENTE HETEROGÊNEO

Trabalho de Conclusão de Curso
apresentado como requisito parcial
à obtenção do grau de Bacharel em
Sistemas de Informação pela
Universidade Feevale

Orientador: Juliano Varella de Carvalho

Novo Hamburgo
2011

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial:

Aos amigos e às pessoas que convivem comigo diariamente, minha gratidão, pelo apoio emocional nos períodos mais difíceis do trabalho.

RESUMO

O desejo de integrar e proporcionar acesso aos dados de um empreendimento é para muitos, uma das principais motivações para o uso de um sistema de banco de dados. A replicação de dados por sua vez, é um processo de compartilhar informações entre diferentes bases de dados. Essa tarefa poderá tornar-se um processo bastante crítico em um ambiente com um número crescente de bases de dados distintas e heterogêneas, e por isto é importante que este processo não afete a desempenho do ambiente ou sistema. Uma replicação mal planejada pode acarretar em riscos para o negócio de uma empresa ou organização. Em algumas situações, como no contexto deste trabalho, pode-se encontrar uma base de dados central. Esta base de dados pode ser caracterizada como uma base com informações sumarizadas, onde se encontram registros replicados a partir de outras bases de dados ou registros de comum interesse para as mesmas. Manter essa base de dados atualizada com as mesmas informações de múltiplas origens não é uma tarefa trivial e pode representar um verdadeiro desafio. Desta forma, este trabalho tem como objetivo propor uma solução aos problemas encontrados na replicação de dados no ambiente de uma empresa.

Palavras-chave: Base de dados. Replicação de dados. Informação. Compartilhar.

ABSTRACT

The desire to integrate and provide access to data of an enterprise is for many people one of the main motivations for using a database system. Data replication in turn, is a process of sharing information among different databases. This task can become a very critical process in an environment with a growing number of different heterogeneous databases, and therefore it is important that this process does not affect the performance of the environment or system. A poorly planned replication can result in risks to a company's business or organization. In some situations, as in the context of this work we have the occurrence of a central database. This database can be characterized as a database with summarized information where can be found replicated records from other databases or records of common interest to them. Keep this database updated with the same information from multiple sources is not a trivial task and may represent a real challenge. Thus, this work aims to propose a solution to the problems encountered in the replication of data in an enterprise environment.

Keywords: Database. Data Replication. Information. Share.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANSI	*American National Standards Institute*
API	*Application Programming Interface*
AW	*Analytical Workspace*
BDD	Banco de Dados Distribuído
BI	*Business Intelligence*
CDC	*Change Data Capture*
CKM	*Check Knowledge Module*
C2F	Commit Duas Fases
C3F	Commit Três Fases
DBLINK	*Database Link*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
DW	*Data Warehouse*
ETL	*Extract Transform Load*
FIFO	*First In First Out*
FTP	*File Transfer Protocol*
GB	*Gigabyte*
GHz	*Gigahertz*
IKM	*Integration Knowledge Module*
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
JDBC	*Java Database Connectivity*
JKM	*Journalizing Knowledge Module*
KMs	*Knowledge Modules*
LCR	*Logical Change Record*
LDAP	*Lightweight Directory Access Protocol*
LKM	*Load Knowledge Module*
MTI	*Multi-table Insert*
ODI	*Oracle Data Integrator*
OGG	*Oracle GoldenGate*
OLAP	*On-line Analytical Processing*

ORA	Oracle
PG	PostgreSQL
PK	*Primary Key*
PL	*Procedural Language*
RKM	*Reverse Knowledge Module*
SCD	*Slowly Changing Dimensions*
SCP	*Secure Copy*
SGBD	Sistema Gerenciador de Banco de Dados
SGBDD	Sistema Gerenciador de Banco de Dados Distribuído
SKM	*Service Knowledge Module*
SOA	*Service Oriented Architecture*
SQL	*Structured Query Language*
SR	*Streaming Replication*
SSO	*Sign-on*
TCP	*Transmission Control Protocol*
WAL	*Write-Ahead Log*
XE	*Express Edition*

SUMÁRIO

INTRODUÇÃO

Devido à rápida expansão da empresa[1] no território nacional, existe a necessidade de compartilhar informações a partir de diversas bases de dados. A empresa em questão atua no segmento de varejo e possui diversas unidades de venda por todo o país. Em cada unidade comercial existe uma base de dados local. E existe em sua unidade administrativa, uma base de dados central, com dados referentes a todas as suas filiais. Devido a este modelo de arquitetura apresentada pela empresa, torna-se necessária a utilização de processos de replicação de dados, proporcionando o compartilhamento de informações entre as unidades da organização, e suas bases de dados.

"Um banco de dados é uma coleção estruturada de dados relacionados a alguns fenômenos reais que estamos tentando modelar." (ÖZSU; VALDURIEZ, 2001, p. 29). Ainda segundo o autor, uma das motivações importantes por trás do uso de sistemas de bancos de dados é o desejo de integrar os dados operacionais de um empreendimento e proporcionar acesso centralizado e, portanto controlado a esses dados.

Para Oracle (2003), replicação de dados significa que os mesmos dados são disponibilizados em diversos locais.

> Replicação é um processo de cópia e manutenção de objetos do banco de dados, como tabelas, nos múltiplos bancos de dados que compõem um sistema distribuído. Alterações aplicadas em um *site* específico são capturadas e armazenadas localmente antes de serem enviadas e aplicadas a cada uma das localizações remotas. (ORACLE, 2003, p.1, tradução nossa).

Um banco de dados replicado não é exatamente um banco de dados distribuído. Neste, os dados estão disponíveis em vários locais, mas uma tabela em particular encontra-se em apenas um local. "Os computadores de um sistema de banco de dados distribuídos recebem diversos nomes, como *sites* ou nós, dependendo do contexto no qual são inseridos." (SILBERCHATZ; KORTH; SUDARSHAN, 1999, p.557).

Entre os benefícios da utilização da replicação de dados destacam-se o desempenho, e a disponibilidade. Segundo Özsu e Valduriez (2001) o uso da replicação de dados pode ser desejável por motivos de desempenho, confiabilidade e disponibilidade. Para Date (2004), a replicação pode ser desejável por duas razões. Primeiramente, ela pode prover um ganho de

---

[1] A empresa em questão não autorizou a divulgação do seu nome, porém, está ciente deste estudo e autorizou o uso das informações contidas nas suas bases de dados.

desempenho, visto que as aplicações podem executar localmente ao invés de terem de se comunicar com um *site* remoto; e em segundo lugar, a replicação pode oferecer uma melhor disponibilidade.

Um banco de dados replicado também apresenta algumas desvantagens. Date (2004) enfatiza que a maior desvantagem da replicação é quando um objeto replicado é alterado, pois todas as cópias do objeto precisam ser atualizadas. "Em geral, a replicação aumenta o desempenho nas operações *read* e aumenta a disponibilidade dos dados para as transações de somente leitura. Entretanto, as transações de atualização geram um grande *overhead*." (SILBERCHATZ ; KORTH; SUDARSHAN 1999, p.590).

No contexto da organização identifica-se a seguinte situação: o cadastro de produtos precisa estar centralizado na matriz, já o faturamento é feito pela unidade, e este não pode ser comprometido em função de alguma falha de comunicação entre a unidade e a matriz. Na matriz é utilizado o SGBD Oracle, porém, em função do custo, nas unidades é utilizado o SGBD PostgreSQL, o que caracteriza um ambiente heterogêneo. Para Elmasri e Navathe (2003), um ambiente onde todos os servidores utilizam o mesmo *software* é chamado de homogêneo.

> Se todos os servidores (ou SGBDs locais) utilizam o mesmo *software* e todos os usuários (clientes) utilizam o mesmo *software*, o SGBDD (Sistema de Gerenciamento de Banco de Dados Distribuído) é chamado de homogêneo, caso contrário é chamado de heterogêneo. (ELMASRI; NAVATHE, 2003, p.815, tradução nossa).

Atualmente a replicação entre as bases de dados é realizada através de um *software* desenvolvido pela própria empresa. Este programa identifica as atualizações ocorridas em uma determinada origem aplicando-as aos demais destinos. Inicialmente este sistema atendia as necessidades da empresa. Porém, com a expansão da própria houve um aumento significativo no volume de dados e consequentemente no volume de replicações.

Com o crescimento no volume de dados replicados a execução deste *software* tornou-se bastante instável. Ocorrem demoras excessivas na atualização das réplicas e sobrecarga no servidor da base de dados central, sendo necessário muitas vezes efetuar uma atualização em lote manualmente em cada base de dados fora do horário de expediente.

Muitas vezes existem dados críticos que devem ser replicados de modo instantâneo. Apesar de o replicador verificar as atualizações ocorridas na base de dados de origem em um curto intervalo de tempo, a atualização das réplicas pode ocorrer com dias de atraso, tornando-se mais um agravante do ambiente. Pois não existe a possibilidade de definir prioridades nas atividades de replicação.

Além disso, não há um controle de falha na atualização de dados em nenhuma das bases se dados. O programa responsável pela atualização das réplicas não controla o *status* de uma atualização, se esta foi executada com sucesso ou falha. Com isso, algumas vezes podem ser encontradas réplicas com estruturas desatualizadas.

Além de garantir a consistência das réplicas, o replicador deve realizar a atualização das réplicas de forma transparente aos usuários do sistema. Considerando que a transparência é uma importante propriedade de bancos de dados distribuídos. É a capacidade de ocultar detalhes da implementação ou distribuição das bases de dados.

Para Silberchatz, Korth e Sudarshan (1999), a transparência é definida como o grau de desconhecimento que os usuários do sistema podem manter em relação ao armazenamento dos dados dentro de um sistema distribuído. Para Özsu e Valduriez (2001), em sistemas de bancos de dados centralizados, o único recurso disponível que precisa ser isolado dos usuários são os dados. Ainda segundo o autor, em um ambiente de bancos de dados distribuídos a rede é um recurso que deve ser administrado da mesma forma, o usuário deve ser protegido contra detalhes operacionais da rede, se possível é desejável, até mesmo, ocultar a existência da rede.

Com base nos conceitos citados, o objetivo deste trabalho é propor uma solução para os problemas no processo de replicação de dados existente, proporcionando melhor escalabilidade e controle para o mesmo, estando ciente de que a empresa encontra-se em uma crescente expansão no mercado nacional, o que representa um aumento no número de bases de dados no cenário de replicação.

Através dos quesitos destacados no parágrafo acima, este trabalho apresenta características e recursos de algumas ferramentas estudadas. São descritos com maior ênfase os recursos e o funcionamento do *Oracle Data Integrator*, que por suas características é a ferramenta que entre as estudadas, é a mais completa em recursos. Pelos recursos apresentados por esta ferramenta, acredita-se que esta possa contemplar todos os requisitos do ambiente, suprindo as carências deste. Solucionando assim as deficiências apresentadas pelo mesmo.

Com o intuito de comprovar a eficácia do *Oracle Data Integrator* no contexto da organização, é realizada uma série de experimentos com a ferramenta, submetendo-a a alguns testes de replicação com algumas estruturas do ambiente. A seguir é abordado brevemente o que será tratado em cada um dos capítulos.

O primeiro capítulo deste trabalho apresenta os conceitos de banco de dados distribuídos, armazenamento de dados e processamento de consultas distribuídas. Apresentando também conceitos de replicação de dados e gerenciadores de réplicas.

No segundo capítulo é descrito o cenário atual da empresa. Descrevendo o processo de replicação e seu funcionamento. Apresentando o gerenciador de replicação como determinado dado é submetido ao processo de atualização das réplicas.

O terceiro capítulo apresenta algumas ferramentas para replicação de dados, dando ênfase a ferramentas com capacidade de replicar dados entre bancos de dados heterogêneos, pois um recurso essencial no contexto da empresa é a replicação entre os SGBDs Oracle e PostgreSQL.

O quarto capítulo consiste na descrição das características e recursos da ferramenta escolhida. Também é descrito o funcionamento desta, a criação de um novo projeto e o desenvolvimento de um módulo de replicação entre bases de dados. Além disso, é caracterizado o ambiente de testes utilizado e os resultados obtidos pelos mesmos. Ainda neste capítulo são apresentados alguns aspectos positivos e negativos identificados durante a etapa de testes com a ferramenta.

1 BANCO DE DADOS DISTRIBUÍDOS

Normalmente, um banco de dados está disponível em apenas um local. Já um banco de dados distribuídos, é formado por um conjunto de bancos de dados que estão distribuídos em vários computadores, e que estão inter-relacionados através de uma rede. Para Coulouris, Dollimore e Kindberg (2007), um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens. Assim sendo, pode-se definir banco de dados distribuídos como uma coleção de vários bancos de dados distribuídos por uma rede de computadores, que estão logicamente interligados. "Um sistema de gerenciamento de banco de dados distribuído (SGBD distribuído) é definido então como o sistema de *software* que permite o gerenciamento do banco de dados distribuído e que torna a distribuição transparente para os usuários." (ÖZSU; VALDURIEZ, 2001, p.5).

Bancos de dados distribuídos podem apresentar algumas desvantagens em relação a um banco de dados convencional, como custo elevado, complexidade de administração e controle de segurança, falta de padrões entre sistemas ou ainda dificuldade no controle de integridade dos dados. No entanto algumas das motivações para utilização de um sistema de banco de dados distribuídos são: o compartilhamento de recursos, e melhor disponibilidade dos dados quando há acesso de aplicações em locais geograficamente distantes.

Para Coulouris, Dollimore e Kindberg (2007), a construção um sistema de bancos de dados distribuídos envolve vários desafios como:

- Heterogeneidade: os sistemas são construídos a partir de *hardware*, e *software* diferentes.
- Sistemas abertos: devem ser capazes de integrar-se com outros sistemas.
- Segurança: utilização de técnicas como criptografia para manter a proteção a informações sigilosas.
- Escalabilidade: o custo da adição de novos usuários deve ser uma constante em relação aos recursos que devem ser adicionados.
- Tratamento de falhas: cada componente deve ser projetado para tratar todas as falhas possíveis.
- Concorrência: cada recurso precisa manter seus estados consistentes perante operações concorrentes.

- Transparência: certos detalhes de implementação devem ser invisíveis aos usuários.

Em um banco de dados distribuído homogêneo todos os *sites* possuem o mesmo SGBD com estruturas (esquemas) idênticas, já um BDD heterogêneo pode possuir não apenas um SGBD distinto como também um esquema diferente dos demais. Se algum *site* de um banco de dados distribuído apresenta um esquema local diferente, torna-se necessária a utilização de um processo de integração. Este processo tem por finalidade integrar o esquema, do banco de dados local para um esquema global do banco de dados distribuído.

De acordo com Ramakrishnan e Gehrke (2008), em um sistema de BDD, os dados são armazenados em diversos *sites*, gerenciados normalmente por um SGBD capaz de executar de forma independente dos outros *sites*. Ainda segundo o autor, o acesso a uma relação armazenada em um *site* remoto acarreta custos de passagem de mensagem e, para reduzir essa sobrecarga podem ser utilizadas algumas técnicas de armazenamento distribuído de dados, como fragmentação e replicação.

Da mesma forma que o armazenamento, o processamento das consultas distribuídas também difere de um processamento local. Pois este deve localizar os fragmentos necessários à execução, identificando aqueles que representam o menor custo para sua execução.

1.1 Armazenamentos distribuídos

O armazenamento de dados em um banco de dados distribuído difere de um banco de dados tradicional. Para Silberchatz, Korth e Sudarshan (1999), um banco de dados distribuído pode ter vários enfoques quanto ao armazenamento de dados como replicação, fragmentação, ou ambas. Em uma arquitetura replicada, o BDD mantém réplicas idênticas das tabelas ou relações em diferentes *sites*. Em um modelo fragmentado, uma relação é dividida em fragmentos que são distribuídos entre *sites* distintos. Cada fragmento corresponde a um subconjunto da relação. A fragmentação pode ser horizontal, vertical ou mista:

- Fragmentação horizontal, uma relação é particionada em subconjunto das linhas da relação original;
- Fragmentação vertical, uma relação é particionada em subconjunto dos atributos (colunas) da relação original;

- Fragmentação mista, também chamada de fragmentação híbrida ou aninhada, caracteriza-se pela utilização conjunta das estratégias de fragmentação horizontal e vertical.

Para Özsu e Valduriez (2001) a fragmentação geralmente resulta na execução paralela de uma consulta, pois a decomposição de uma relação em fragmentos permite que várias transações sejam executadas de forma concorrente, aumentando o nível de concorrência e, por conseguinte a vazão do sistema. Além disso, a fragmentação de dados pode reduzir efeitos negativos da replicação, pois exige menos espaço em disco, e também, menos itens de dados precisam ser administrados. No entanto, como desvantagem pode-se mencionar a verificação de integridade dos dados, pois estruturas muito fragmentadas podem, por exemplo, resultar em consultas a diversos *sites* para efetuar uma verificação de dependência.

Özsu e Valduriez (2001) enfatizam que, em sistemas de bancos de dados centralizados, o único recurso disponível que precisa ser isolado dos usuários são os dados. Ainda segundo o autor, em um ambiente de bancos de dados distribuídos a rede é um recurso que deve ser administrado da mesma forma, o usuário deve ser protegido contra detalhes operacionais da rede, se possível é desejável, até mesmo, ocultar a existência da rede. Em outras palavras, as operações de acesso aos dados de um objeto devem ser transparentes ao usuário. Ou seja, o acesso a uma relação é idêntica, tanto para objetos locais como para objetos remotos.

1.1.1 Transparência

A transparência é uma importante propriedade de bancos de dados distribuídos. Autores como Özsu e Valduriez (2001), Coulouris, Dollimore e Kindberg (2007) e Silberchatz, Korth e Sudarshan (1999) descrevem a transparência de um sistema como sendo a capacidade do sistema de ocultar os detalhes de implementação.

> A transparência é definida como sendo a ocultação, para um usuário final ou para um programador de aplicativos, da separação dos componentes em um sistema distribuído de modo que o sistema seja percebido como um todo, em vez de uma coleção de componentes independentes. (COULOURIS; DOLLIMORE; KINDBERG, 2007 p.34-35).

Segundo ANSA (1989 apud Coulouris, Dollimore e Kindberg, 2007), e o ISO (1992 apud Coulouris, Dollimore e Kindberg, 2007), identificam oito formas de transparência:

- Transparência de acesso: recursos locais e remotos são acessados de forma idêntica.
- Transparência de localização: recursos são acessados sem conhecimento de sua localização física.
- Transparência de concorrência: vários processos operam concorrentemente, utilizando recursos compartilhados sem interferência entre os mesmos.
- Transparência de replicação: várias instâncias dos recursos são utilizadas para aumentar a confiabilidade e o desempenho, sem conhecimento das réplicas por parte dos usuários.
- Transparência de falhas: garante que usuários e programas aplicativos concluam suas tarefas, a despeito de alguma falha.
- Transparência de mobilidade: proporciona a movimentação de recursos e clientes dentro de um sistema, sem afetar a operação dos usuários ou programas.
- Transparência de desempenho: permite que o sistema seja reconfigurado para melhorar o desempenho à medida que as cargas variam;
- Transparência de escalabilidade: garante que o sistema se expanda em escala, sem alterar a estrutura do sistema ou os algoritmos de aplicativo.

Coulouris, Dollimore e Kindberg (2007) afirma que o meio para ser obter transparência de replicação, é a utilização de um componente chamado *front end*, cuja função é comunicar-se com um ou mais gerenciadores de réplicas por meio de passagem de mensagens, sem que seja necessário ao usuário tomar conhecimento da replicação.

1.1.2 Gerenciadores de réplica

Um gerenciador de réplica é responsável pela atualização dos *sites*, tornando o processo transparente ao usuário. O usuário utiliza um objeto ou tabela do sistema, que na verdade pode estar sendo replicada pelo gerenciador de replicação. Um conjunto de gerenciadores de réplica pode ser estático ou dinâmico. Coulouris, Dollimore e Kindberg (2007) enfatizam que em um sistema dinâmico, novos gerenciadores de réplica podem surgir a qualquer momento, como por exemplo, uma segunda secretária copia uma agenda em seu *laptop*, o que em um sistema estático não é permitido. “Em um sistema dinâmico, os gerenciadores de réplica podem falhar e, então, eles são considerados como tendo deixado o sistema (embora possam ser substituídos)” (COULOURIS; DOLLIMORE; KINDBERG,

2007, p.524). “Em um sistema estático os gerenciadores de réplica não falham (a falha implica em nunca executar outro passo), mas deixam de funcionar por um período indefinido“ (COULOURIS; DOLLIMORE; KINDBERG, 2007, p.524).

Ao executar operações sobre réplicas, um gerenciador de replicação deve ser capaz de fazê-las de forma recuperável, de maneira a não permitir inconsistências na ocorrência de alguma falha durante o processo em alguma das réplicas. Este deve também decidir a ordem de execução das operações. Coulouris, Dollimore e Kindberg (2007) apresentam três tipos de ordenação: Ordenação FIFO, causal e ordenação causal.

> Ordem FIFO: se um *front end* emite uma requisição r e depois uma r’, então um gerenciador de réplica correto que trate r’, processará r antes dela.
> Ordem causal: se a emissão da requisição r aconteceu antes da emissão de r’, então um gerenciador de réplica correto que trate r’, processará r antes dela.
> Ordem total: se um gerenciador de réplica correta tratar r antes de solicitar r’, então um gerenciador de réplica correto que trate r’, processará r antes dela. (COULOURIS; DOLLIMORE; KINDBERG, 2007, p.524)

Gerenciar a ordem de execução das atualizações é um processo fundamental em um sistema replicado, pois podem existir dependências de dados entre as modificações que ainda não foram propagadas. Um gerenciador de réplicas deve ser capaz de identificar ou tratar de modo adequado essas dependências com o intuito de que estas não resultem em uma falha de atualização.

Uma dependência pode ser representada pela relação entre duas tabelas A e B. A tabela B está relacionada à tabela A através de uma chave estrangeira. No instante que as modificações forem propagadas para as réplicas, o gerenciador de replicação precisa garantir que a atualização da tabela A na réplica ocorrerá antes da tabela B, caso contrário a atualização da tabela B poderá resultar em uma falha.

1.1.3 Tolerância a falhas

Todo sistema está sujeito a falhas. Um sistema é caracterizado como tolerante a falhas quando este é capaz de manter o seu comportamento de acordo com a sua especificação, mesmo na ocorrência de um imprevisto. De acordo com Özsu e Valduriez (2001), um SGBD distribuído confiável deve ser capaz de continuar a executar solicitações do usuário sem violar a consistência do banco de dados, mesmo quando componentes do ambiente de computação distribuída falham.

Existem diversos tipos de falhas como àquelas de transação, comunicação ou de *hardware*. Falhas de transação podem ocorrer devido a algum erro lógico no sistema, ou por algum erro no sistema como um *deadlock*, enquanto as falhas de *hardware* podem ocorrer

devido a problemas com componentes físicos. Para Özsu e Valduriez (2001), é considerada uma falha, qualquer divergência de um sistema em relação ao comportamento descrito na especificação, enquanto que um defeito do sistema é qualquer erro nos estados internos dos componentes, ou no projeto do mesmo.

Para Silberchatz, Korth e Sudarshan (1999), um sistema distribuído está sujeito aos mesmos tipos de falhas de um sistema centralizado (como por exemplo, erros de *software*, *hardware* ou erros em disco). No entanto, há necessidade de que em um ambiente distribuído alguns tipos adicionais sejam devidamente tratados. Algumas situações características são: falha de comunicação ou em *sites;* perda de mensagens; e problemas de particionamento da rede.

Quando ocorre uma falha no SGBD, este deve ser capaz de retornar ao seu estado anterior à ocorrência do imprevisto. Para isso, é necessário que o SGBD sempre mantenha informações sobre o seu estado. Essas são chamadas de informações de recuperação. "Uma parte integrante de um sistema de banco de dados é o esquema de recuperação que é responsável pela restauração do banco de dados para um estado consistente que havia antes da ocorrência da falha." (SILBERCHATZ; KORTH; SUDARSHAN, 1999, p.511).

Özsu e Valduriez (2001) e Silberchatz, Korth e Sudarshan (1999) afirmam que o banco de dados deve manter seu estado consistente. Mesmo na ocorrência de um evento inesperado o banco de dados deve garantir a atomicidade e a durabilidade das transações. A confiabilidade se refere tanto a resiliência de um sistema a vários tipos de falhas quanto à sua capacidade de se recuperar-se das mesmas. "Um sistema resiliente é tolerante às falhas do sistema e pode continuar a fornecer serviços, mesmo quando ocorrem falhas." (ÖZSU; VALDURIEZ, 2001, p.292).

Özsu e Valduriez (2001) enfatizam que a confiabilidade de um banco de dados distribuídos é assegurada pelos protocolos de consolidação, término e recuperação. Ainda de acordo com o autor, o principal requisito destes protocolos de consolidação é que mantenham a atomicidade de transações distribuídas. Isso significa que, embora a execução da transação distribuída envolva vários *sites*, alguns dos quais podem falhar durante a execução. "O efeito da transação sobre o banco de dados distribuído é tudo ou nada. Isso se chama consolidação atômica." (ÖZSU; VALDURIEZ, 2001, p.406).

1.1.4 Transações distribuídas

Para Özsu e Valduriez (2001), uma transação é uma unidade de computação consistente e confiável, que é constituída por uma sequência de operações de leitura e escrita sobre o banco de dados. O gerenciamento de transações lida com os problemas de sempre manter a base em um estado consistente, mesmo quando ocorrem acessos e falhas concorrentes. Ainda segundo o autor, um banco de dados está em um estado consistente se obedece a todas as restrições de consistência (integridade) definidas sobre ele, enquanto que um banco replicado encontra-se em um estado mutuamente consistente, se todas as cópias de todos os itens de dados que ele contém apresentam valores idênticos.

"Se a transação pode completar sua tarefa com sucesso, dizemos que a transação se consolida. Por outro lado, se a transação pára sem completar sua tarefa, dizemos que ela aborta." (ÖZSU; VALDURIEZ, 2001, p.296).

Uma transação para ser consistente e confiável depende de propriedades como atomicidade, consistência, isolamento e durabilidade. A atomicidade, também conhecida como propriedade "tudo ou nada" garante que uma transação só é concluída quando todas as suas ações forem concluídas. A consistência corresponde à integridade dos dados. Ela assegura que as regras do banco de dados não sejam quebradas. O isolamento garante que transações concorrentes não revelem seus estados antes de consolidar. A durabilidade assegura que os dados, uma vez consolidados, tornam-se permanentes no banco de dados.

Para Silberchatz, Korth e Sudarshan (1999), em um sistema distribuído existem dois tipos de transações que devem ser consideradas, as transações locais que atualizam apenas a base local, e as transações globais que atualizam diversas bases de dados. Para o autor, cada sistema de transações possui dois subsistemas: o gerenciador de transações que administra transações no *site* local, e o coordenador de transações, responsável por coordenar a execução de diversas transações, tanto locais como globais.

Para garantir a atomicidade em um banco de dados distribuído, torna-se necessária a utilização de um protocolo de consolidação, que assegure que todos os *sites* envolvidos concordem com o término da transação, em outras palavras é preciso que a transação seja consolidada em todos os *sites*, ou abortada por todos eles.

1.1.5 Protocolos de consolidação

O protocolo de consolidação em duas fases (C2F) é um dos mais simples e utilizados protocolos que assegura a consolidação atômica de transações distribuídas. Seu nome origina-se do fato que ocorrem duas rodadas de mensagens. A primeira de votação e a segunda fase

de finalização. As duas são iniciadas pelo coordenador da transação, e cada mensagem enviada sinaliza uma ação tomada pelo remetente.

"O gerenciador de transações no *site* onde a transação originou é chamado de coordenador da transação; os gerenciadores de transação nos *sites* onde suas subtransações são executadas são chamados de subordinados". (RAMAKRISHNAN; GEHRKE, 2008 p. 630).

Neste protocolo, todos os participantes devem chegar a um consenso para consolidar uma transação. "Ele estende os efeitos de ações locais de consolidação atômica e transações distribuídas, insistindo que todos os *sites* envolvidos na execução de uma transação distribuída concordem em consolidar a transação, antes de seus efeitos se tornarem permanentes." (SILBERCHATZ; KORTH; SUDARSHAN, 1999, p.406)

Em um protocolo C2F, o coordenador da transação envia uma mensagem *prepare* para cada participante. Cada subordinado deve decidir então se cancela ou consolida a sua subtransação, e então deve enviar uma mensagem de resposta ao coordenador da transação. *vote-commit* para consolidar ou *vote-abort* para cancelar a transação.

Um coordenador, ao receber as mensagens de todos os seus subordinados, decide então por consolidar ou cancelar a transação global. Caso um participante envia uma mensagem de *vote-abort* o coordenador deverá abortar a transação de forma global, enviando uma mensagem de *global-abort* a todos os participantes.

Caso ocorra uma falha no coordenador da transação, os participantes não podem decidir se vão cancelar ou consolidar a sua subtransação, até que o *site* coordenador se recupere. Mantendo assim a transação em estado de bloqueio.

O protocolo consolidação em três fases pode evitar o problema de bloqueio do C2F. Neste protocolo o coordenador adia a decisão de consolidar a transação até que participantes suficientes saibam sobre a decisão de consolidar. Neste caso, se houver uma falha no coordenador, estes participantes poderão se conectar e identificar que a transação deve ser consolidada.

A grande desvantagem do modelo C3F para o C3F é que o protocolo de três fases apresenta maior custo durante a execução. "O protocolo C3F impõe um custo adicional significativo durante a execução normal e exige que falhas de *link* de comunicação não levem a uma partição de rede para garantir liberdade de bloqueio." (RAMAKRISHNAN; GEHRKE, 2008 p. 634).

Para Ramakrishnan e Gehrke (2008), a transação está oficialmente efetivada quando o registro de *log* de consolidação do coordenador chega ao armazenamento estável. Isso significa que falhas posteriores não afetam o resultado da transação.

1.2 Replicação de dados

A replicação pode ser aplicada não apenas às modificações em dados das tabelas de um sistema (alterações DML), como também às modificações dos próprios objetos de um banco de dados (alterações DDL) como tabelas, visões sequências e outros. A maioria das ferramentas de replicação implementa apenas replicação de dados de tabelas, e não estruturas de objetos. No entanto, existem ferramentas alternativas que possibilitam a execução de um comando DDL em todas as réplicas, ou nós de um sistema distribuído.

> As réplicas são objetos físicos, armazenados em locais distintos, que são manipulados pelo sistema através de um objeto lógico. As réplicas de determinado objeto não são necessariamente idênticas, pelo menos não em um ponto em particular do tempo. Algumas réplicas podem ter recebido atualizações que outras não receberam. (COULOURIS; DOLLIMORE; KINDBERG, 2007, p.523).

Um banco de dados replicado precisa garantir que um objeto quando atualizado, seja atualizado também em todas as réplicas. Garantindo a consistência mútua das bases de dados. Esta replicação pode ser classificada quanto ao seu modo de execução da seguinte forma:

- A Replicação *eager* ou síncrona: garante que todas as réplicas serão idênticas a cópia primária. Os dados são replicados imediatamente após serem atualizados no servidor. Nesse modo, a transação não é concluída até que todas as réplicas sejam atualizadas. Uma operação de *rollback* reverte os dados de todas as cópias, garantindo assim, que todas elas sejam idênticas. O problema dessa replicação é que se uma das bases estiver inacessível, nenhuma transação será concluída até que a conexão seja restabelecida.
- Replicação *lazy* ou assíncrona: nesse modo as atualizações das réplicas são efetuadas em uma nova transação, podendo levar algum tempo até serem aplicadas. Essas atualizações também são conhecidas como transações adiadas ou *deferred transactions*. As vantagens do modo assíncrono são o baixo custo com relação ao modo síncrono, a alta escalabilidade e a resistência a quedas de rede.

A replicação também pode ser classificada quanto a sua forma de atualização:

- Replicação *master-slave* ou unidirecional: significa que a atualização das réplicas é feita de modo unidirecional, apenas a base *master* recebe modificações, enquanto as réplicas são utilizadas como apenas leitura. Este modo é normalmente usado para

backups de servidores de bancos de dados ou também para melhoria no desempenho de consultas em *sites* remotos.

- Replicação *multi-master* ou bidirecional: neste modo todas as bases podem sofrer modificações e estas devem ser atualizadas para as demais réplicas. Este modo é bastante utilizado para garantir alta disponibilidade. Nesse modo de replicação podem ocorrer conflitos de dados, pois existe a possibilidade de que uma informação esteja sendo atualizada a partir de mais de um local ao mesmo tempo.

As formas mais comuns para capturar as mudanças ocorridas em uma origem de dados são o uso de *triggers* ou uso de *logs* de transações. A utilização de *triggers* provê uma melhor flexibilidade no momento de decidir quais os dados devem ser replicados. No entanto, o *log* de transações tem como vantagem o baixo custo causado no servidor de dados em função de que este mecanismo lê o *log* em memória, não concorrendo com recursos do servidor de dados.

Em alguns ambientes, pode existir a necessidade de um *software* que transmita as mudanças entre as bases de dados. Isso ocorre geralmente em ambientes que contenham servidores de dados distintos, onde não é possível efetuar uma conexão direta entre as bases de dados, sendo necessário que o *software* capture os registros de mudanças em uma base de dados de origem, propagando-as para todas as bases de destino. Para isto, podem ser utilizados aplicativos de ETL (*Extract Transformation Load*).

1.2.1 Propagação do *update*

Para Date (2004), o principal problema da replicação de dados é que no momento que um objeto sofre uma alteração, todas as suas réplicas precisam ser atualizadas. Ainda segundo o autor, a solução mais óbvia seria atualizar as réplicas imediatamente após a atualização da origem. No entanto, este tipo de operação pode resultar em uma falha de transação, caso uma das réplicas não esteja disponível.

Date (2004), ainda afirma que algumas ferramentas de replicação existentes, executam a replicação em modo assíncrono, propagando as atualizações às replicas após uma operação de *commit* na origem dos dados, o que ocasionalmente pode deixar inconsistências nas réplicas. O autor enfatiza que uma das razões de muitas ferramentas optarem por esse modo de atualização é que, a replicação em modo síncrono exigiria um protocolo de consolidação em duas fases, o que causaria um custo no desempenho da aplicação.

1.3 Processamento de consultas

"Os processamentos de consultas são as atividades envolvidas em extrair dados de um banco de dados." (SILBERCHATZ; KORTH; SUDARSHAN, 1999, p.381). Ainda segundo o autor, em sistemas centralizados, o primeiro critério para mensuração do custo de uma estratégia em particular é o número de acessos a disco. Enquanto nos sistemas distribuídos, devem ser considerados outros problemas, como o custo de transmissão de dados na rede e o ganho de desempenho diante do fato de que diversos *sites* podem processar partes da consulta em paralelo. O custo relativo de transferência de dados na rede e de transferência de dados entre discos varia significativamente, dependendo do tipo de rede e da velocidade dos discos. "Assim, não se pode focalizar somente os custos relativos a discos ou a rede. Deve-se achar o melhor *tradeoff*." (SILBERCHATZ; KORTH; SUDARSHAN, 1999, p.598).

Quando uma consulta é efetuada sobre objetos de dados fragmentados torna-se necessário uma conversão da consulta global para consultas de vários fragmentos. Apesar de uma consulta ser especificada sobre uma relação inteira, é necessário que esta seja mapeada em consultas sobre os fragmentos físicos que compõem a relação. De acordo com Özsu e Valduriez (2001), em um contexto centralizado, as estratégias de execução de consultas podem ser expressas em uma extensão de álgebra relacional, já em um sistema distribuído, a álgebra relacional não é suficiente para expressar as estratégias de execução, e precisa ser complementada com operações para intercâmbio de dados entre *sites*. "Além da opção de ordenar operações da álgebra relacional, o processador de consultas distribuídas deve selecionar os melhores *sites* para processar os dados e, possivelmente, o modo como os dados devem ser transformados." (ÖZSU; VALDURIEZ, 2001, p.202).

2 ESTRUTURA ATUAL DE REPLICAÇÃO

A arquitetura da empresa é formada pelos servidores Oracle e PostgreSQL. Cada loja possui um servidor de dados PostgreSQL local, enquanto a matriz da empresa possui um servidor de dados Oracle. O servidor de dados das lojas armazena informações referentes à unidade. A Figura 2.1 representa a arquitetura *multi-master* de replicação entre as bases de dados. Cada um dos servidores da loja comunica-se com o servidor de dados da matriz, que é uma base de dados global, com dados referentes à matriz da empresa e todas as demais lojas do grupo empresarial.

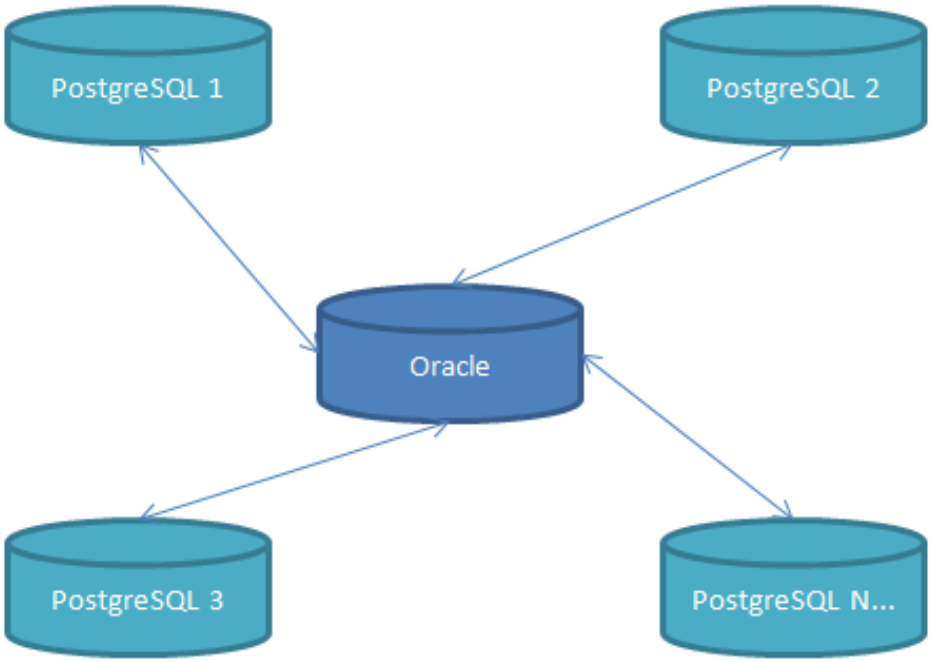

Figura 2.1 - Distribuição das bases de dados

Nem todas as informações das bases de dados precisam ser replicadas, as principais estruturas que devem ser replicadas são referentes aos cadastros. Outras como, por exemplo, estruturas de orçamentos são relevantes apenas na unidade em questão sendo desnecessária sua replicação para a base de dados global. Importante ressaltar que os esquemas são idênticos, independente do SGBD. Objetos como tabelas, visões apresentam as mesmas colunas e tipos de dados em todas as réplicas.

O modo de replicação pode variar de acordo com a estrutura. Algumas estruturas podem ser replicadas de modo unidirecional por serem utilizadas como apenas leitura na base

de dados destino, porém outras devem ser replicadas de modo bidirecional, pois estão sujeitas a alterações em qualquer uma das bases de dados. O cadastro de produtos é gerenciado apenas pela matriz, onde existem pessoas específicas para esta operação e nas lojas é utilizado apenas para consulta, não são efetuadas alterações no cadastro, por isso pode ser replicado de modo unidirecional. Já o cadastro do cliente, pode sofrer alterações tanto na loja como na própria matriz, por isso devem ser replicadas de modo bidirecional, alterações efetuadas na loja devem estar disponíveis na matriz, e vice-versa.

A utilização desse modelo de replicação foi concebida com o intuito de garantir alta disponibilidade às lojas da empresa. Com isso buscou-se evitar que a empresa tenha suas operações normais de trabalho prejudicadas por algum problema de comunicação com a base de dados da matriz. Dessa maneira também foi possível obter melhor desempenho do sistema nas lojas, pois ao invés de acessar um servidor de dados remoto, a aplicação acessa um servidor de dados local.

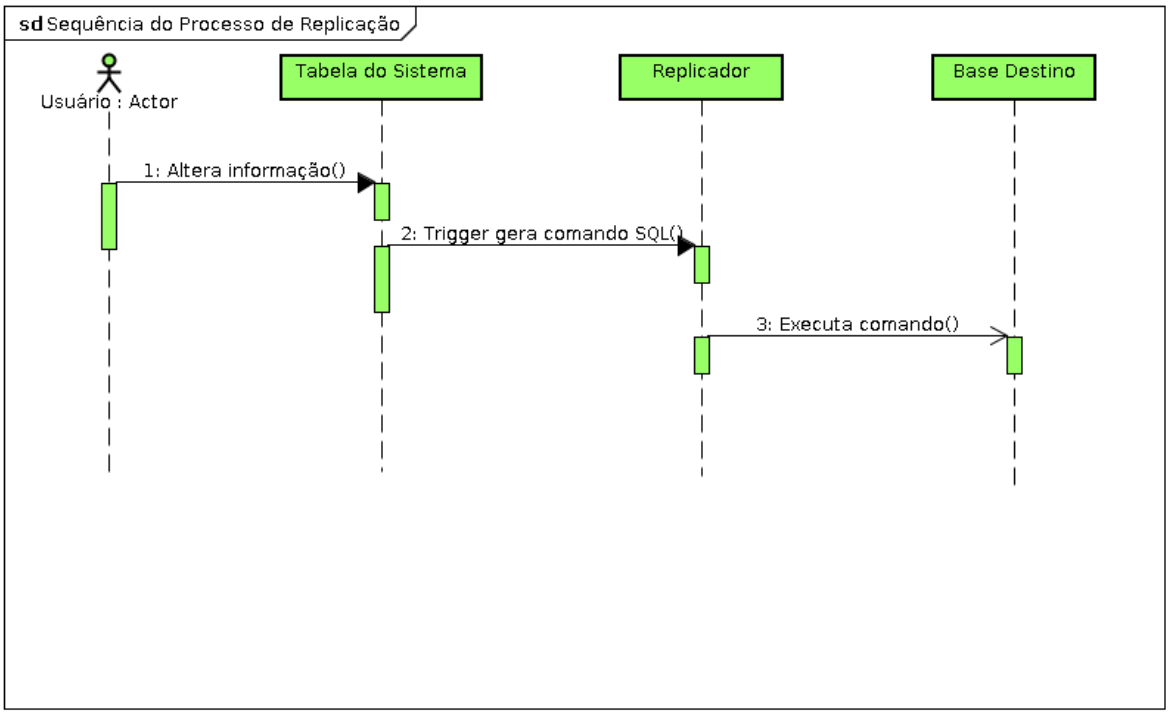

Figura 2.2 - Sequência do processo de replicação

A Figura 2.2 representa o fluxo do processo de replicação, a partir do instante que o usuário altera uma informação no sistema até o momento que essa informação é atualizada à base destino. Para que uma estrutura seja replicada, é necessário criar no banco de dados de origem uma *trigger* responsável pela geração do comando que será aplicado na base de dados destino. Essa *trigger*, comumente executada após os eventos *insert*, *update*, *delete* deve

identificar todos os campos que necessitam ser replicados; gerar o comando DML (que deve ser compatível com o SGBD de destino) responsável pela atualização do destino; e gravar esse comando na tabela de comandos do replicador juntamente com o código do estabelecimento que receberá essa atualização. Uma vez que o comando seja gravado na tabela, o gerenciador de replicação identificará esse comando e se encarregará de efetuar uma conexão com a base de dados destino e executar o comando SQL gerado.

Quando ocorre uma alteração de dados em uma tabela de origem, todas as colunas são atualizadas nas réplicas. Não existe no ambiente, atualização parcial de uma tabela, mesmo que apenas uma coluna seja modificada na origem, todos os campos das réplicas receberão a atualização. A atualização parcial pode ocorrer de forma indesejada se houver uma alteração DDL em uma determinada tabela e não ocorrer à devida alteração na *trigger* correspondente. Por exemplo, adicionando-se a coluna X a tabela Y, sem a customização da *trigger* de replicação esta coluna não será atualizada nas réplicas.

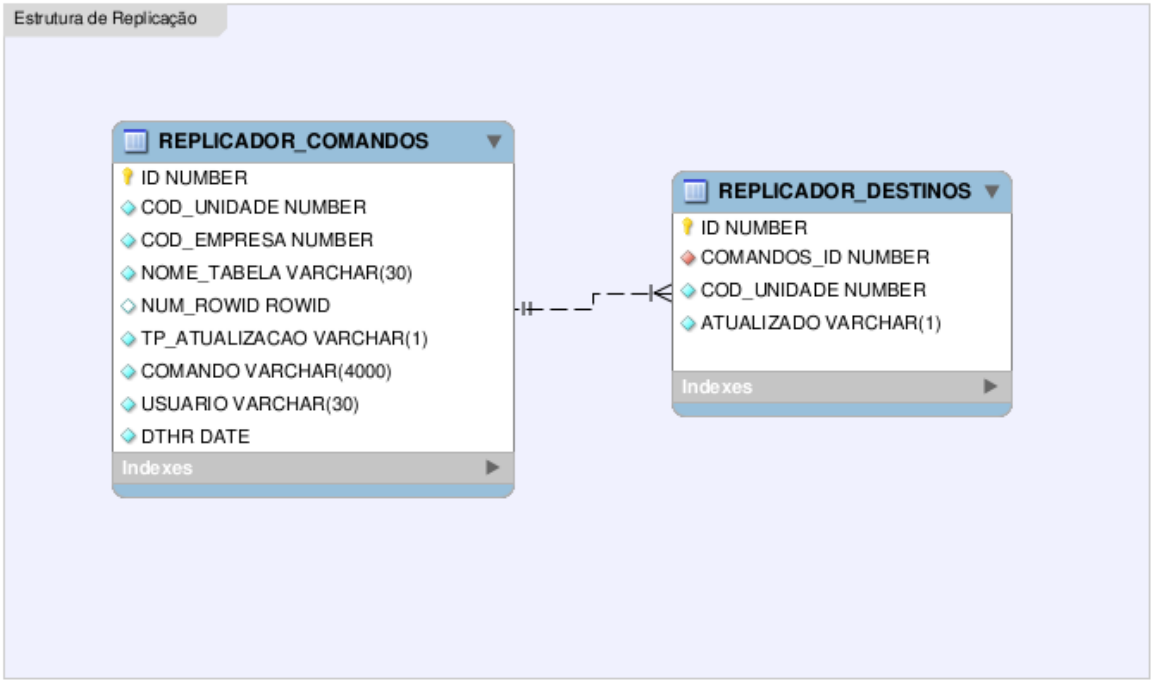

Figura 2.3 - Estrutura de tabelas do replicador

A Figura 2.3 se refere à estrutura da tabela do replicador, a tabela REPLICADOR_COMANDOS, contém os comandos DML gerados pela base de dados de origem, enquanto a tabela REPLICADOR_DESTINOS refere-se às unidades que receberão o comando de atualização pelo gerenciador de replicação. Na tabela REPLICADOR_COMANDOS, os campos COD_UNIDADE e COD_EMPRESA referem-se

à unidade de origem onde o usuário efetuou a alteração da informação. A coluna NOME_TABELA contém o nome da tabela que sofreu atualização, enquanto o TP_ATUALIZACAO indica o tipo de alteração efetuado nessa tabela, que pode ser uma inserção, atualização ou exclusão. O campo NUM_ROWID contém o identificador do registro, e é utilizado apenas na base de dados Oracle, já a coluna comando, contém o comando SQL propriamente dito. Os campos USUARIO e DTHR são referentes ao usuário que efetuou a alteração no sistema, e a data e hora da operação.

A tabela REPLICADOR_DESTINOS contém apenas o código da unidade que receberá o comando de atualização, e um indicador, para o reconhecimento se o comando já foi executado.

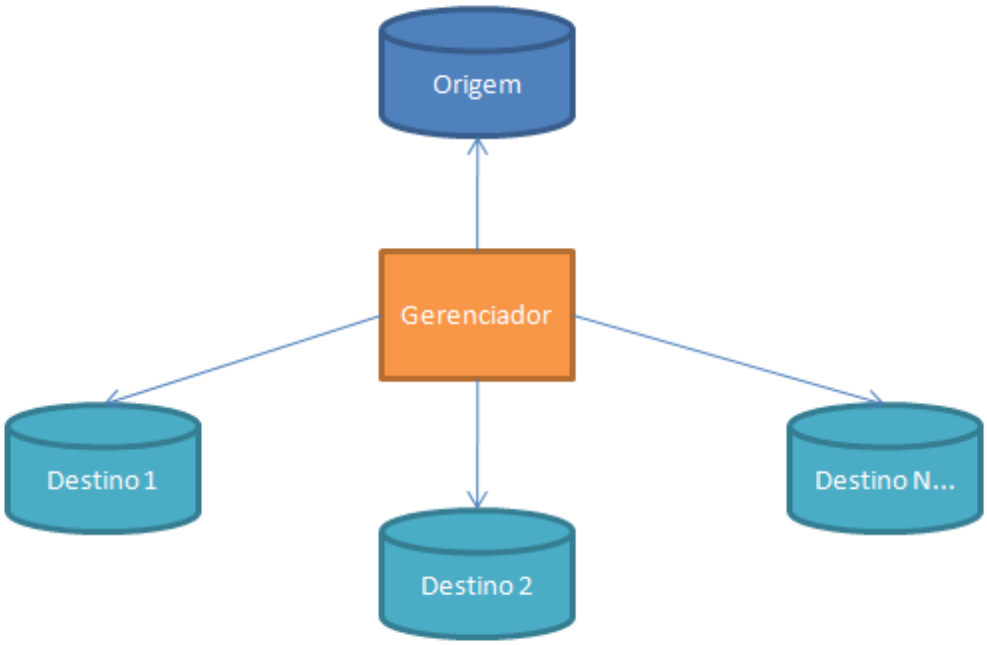

Figura 2.4 - Modelo de comunicação do gerenciador de replicação

O gerenciador de replicação foi desenvolvido internamente, com a ferramenta Visual Basic 6. Sua execução funciona da seguinte forma: a cada intervalo de três minutos, ele verifica novos comandos na tabela REPLICADOR_COMANDOS de uma base de dados de origem e os adiciona em sua lista de execução. De modo assíncrono, cada um dos comandos em sua lista é executado na base de dados destino. Uma instância do gerenciador pode conectar-se a várias réplicas, como mostra a Figura 2.4. O mesmo ainda pode ser parametrizado para verificar atualizações para apenas uma unidade específica. Esse recurso

tem sido comumente utilizado, pois uma vez que o replicador adiciona um comando a sua lista de execução, ele fica em modo de espera, até que todos os comandos da fila de execução tenham sido executados. Deste modo, a vantagem do uso de um gerenciador de replicação por unidade é que cada unidade terá a sua própria fila de execução, com a utilização paralela dos replicadores para cada unidade, o processo será concluído em menor tempo de execução.

2.1 Problemas da estrutura

Para algumas estruturas do banco de dados como, por exemplo, a tabela de clientes, que possui aproximadamente 250 colunas, são necessários no mínimo três comandos DML para atualização das réplicas de uma única tabela. Isso se deve em função da tabela de comandos do replicador possuir uma limitação no número de caracteres do comando, atualmente 4000 *bytes*. Então a cada atualização na tabela de clientes, são gerados mais três novos comandos no replicador para a atualização de cada réplica. Assim sendo, uma alteração na tabela de clientes, considerando um número de cem lojas, resultam em trezentos novos comandos SQL ao replicador. Considerando que, ao cadastrar um novo cliente, existem outras tabelas relacionadas ao cliente, como estruturas de informações financeiras, de endereçamento e outras, o custo dessa atualização será ainda maior.

O grande volume de instruções geradas para replicação, em função das diversas estruturas que devem ser replicadas resulta numa grande carga de trabalho ao servidor de dados central. Como todas as instruções são organizadas em um sistema de filas, a base central poderá conter dados desatualizados enquanto o gerenciador de replicação não finalizar a execução de todos os comandos da fila.

Outro grande problema do gerenciador de replicação é o controle de atualização dos comandos. Algumas vezes ocorrem falhas na atualização em alguma das bases de dados. No entanto, independentemente deste comando ter sido executado com sucesso ou não, o gerenciador atualiza o estado desse comando, indicando que o mesmo foi corretamente executado. Ou seja, não há controle de integridade ou consistência das réplicas. Ocasionalmente encontram-se réplicas desatualizadas, pois no momento da atualização da mesma houve alguma falha na execução do comando na base de dados, que não foi corretamente tratada pelo gerenciador de replicação.

Implantar replicação em uma nova base de dados não é uma tarefa simples. A falta de documentação do ambiente dificulta ainda mais essa tarefa. Para executar essa atividade é

necessário certificar-se da existência de todas as *triggers* responsáveis pela geração dos comandos SQL. Além disso, deve ser configurado e disponibilizado acesso nos gerenciadores de réplica às bases de dados.

Quando há alguma mudança de estrutura em tabelas do banco de dados é necessário modificar a *trigger* correspondente para replicação da mesma. Uma vez desenvolvida e homologada a alteração desta rotina, torna-se necessário atualizar este objeto em todos os bancos de dados. Como esses objetos não possuem controle de versão, o gerenciamento de versões é feito de forma manual. Para certificar-se de que todos os objetos de todas as bases estão iguais é necessário utilizar algum *software* para comparação de texto. Comparando o código-fonte das mesmas em todas as bases de dados.

As dificuldades descritas caracterizam as limitações do gerenciador e estrutura de replicação. Para obter um ambiente estável, de fácil gerenciamento e manutenção, torna-se necessário a utilização de uma ferramenta distinta para gerenciamento das réplicas. Atualmente existem diversas ferramentas para replicação de dados na *internet.* No entanto, a maioria delas apresenta diversas limitações, como pôde ser constatado através da análise de algumas dessas ferramentas de replicação, as quais são abordadas no próximo capítulo.

3 FERRAMENTAS DE REPLICAÇÃO DE DADOS

Existem diversas ferramentas disponíveis para desenvolvimento de processos para replicação de dados. Para este estudo foram pesquisadas algumas ferramentas que oferecem suporte a replicação de dados no Oracle e no PostgreSQL.

3.1 Replicação de dados no PostgreSQL

O PostgreSQL, até versões anteriores a 9.0, não contava com nenhum recurso nativo para replicação de dados, sendo necessária a instalação de complementos ou ferramentas de terceiros para realização deste processo. Um dos complementos mais utilizados para replicação de dados no PostgreSQL é o Slony-I.

Existem diversas ferramentas de terceiros disponíveis no mercado para replicação de dados com PostgreSQL, como por exemplo, Postgres-R[2], e o PG-Replicator[3] que não serão abordadas neste estudo em função de que estas ferramentas estão limitadas ao uso do PostgreSQL, sendo que o objeto deste estudo é a replicação entre Oracle e PostgreSQL.

3.1.1 Slony-I

O Slony-I é uma ferramenta multiplataforma para replicação de dados entre servidores PostgreSQL. De acordo com a documentação do Slony-Info (2009), o sistema de replicação do Slony-I baseia-se na utilização de *triggers* para verificação de mudanças ocorridas na base de dados. De acordo com Wieck (2010), as *triggers* do Slony-I, são disparadas no evento AFTER ROW de uma tabela. Outra característica do Slony-I é que colunas não modificadas não são atualizadas em uma cláusula UPDATE durante a atualização da réplica.

Com esse sistema de verificação surgem algumas limitações como alterações DDL de objetos, ou de usuários e *roles* não podem ser replicadas. Para replicação de comandos DDL, é possível utilizar o SLONIK, um utilitário que permite executar comandos SQL em todos os nós ou réplicas previamente informados.

---

[2] Disponível em: < http://postgres-r.org/> Acesso em: 10 out. 2010.

[3] Disponível em: < http://pgreplicator.sourceforge.net/> Acesso em: 20 out. 2010.

Segundo Slony-Info (2009), para cada banco de dados do sistema é iniciado um serviço chamado *slon*, este é responsável por processar os eventos de replicação. Ainda de acordo com Slony-Info (2009), estes eventos podem ser de configuração ou eventos SYNC. Eventos de configuração geralmente ocorrem na execução de *scripts* com o SLONIK que executam alterações na configuração de um *cluster*. Eventos SYNC são atualizações de tabelas que são agrupadas em grupos de transação.

De acordo com Slony-Info (2010), a ferramenta Slony-I apresenta algumas vantagens e também algumas limitações quanto ao processo de replicação de dados.

3.1.1.1 Vantagens do Slony-I

- Pode replicar dados entre diferentes versões do PostgreSQL.
- Possibilita determinar quais tabelas devem ser replicadas.
- Possibilita definir diferentes servidores como origem ou destino de diferentes tabelas.
- Possui um recurso de atualização das réplicas em cascata. Ou seja, as réplicas não precisam receber os dados diretamente da base *master*. Uma réplica após receber a atualização da base *master* pode encaminhar a atualização para outra réplica. Este recurso é chamado *cascaded slaves*.

3.1.1.2 Limitações e desvantagens do Slony-I

- Estrutura homogênea, ou seja, as estruturas de origem e destino devem ser idênticas;
- Replicação apenas de tabelas e sequências. Demais objetos devem ser importados pelo usuário;
- Slony-I deve ser instalado em cada servidor, *master* ou *slave*;
- Alterações de esquema e *large object* não podem ser capturadas por *triggers*;
- As bases de dados devem possuir o mesmo *encoding*;

3.1.2 PostgreSQL *streams*

Recentemente lançada, a versão 9.0 do PostgreSQL traz uma recurso próprio para replicação entre bases de dados. O recurso de replicação com *streams*. Este mecanismo possibilita a replicação assíncrona entre bases de dados PostgreSQL. Uma grande vantagem da replicação com *streams* é a baixa carga que este impacta no servidor. De acordo com

PostgreSQL (2010a), este novo recurso apresenta ainda algumas limitações quanto a sua utilização.

- Não há suporte para replicação entre SGBDs distintos.
- Cada *site* deve estar rodando com a mesma versão do PostgreSQL, sob a mesma plataforma;
- A estrutura dos servidores deve ser idêntica, um servidor não pode ter tabelas ou banco de dados extras, até mesmo os índices das tabelas devem ser idênticos;
- Não é possível replicar apenas parte das alterações que estiverem ocorrendo no banco de dados. A replicação com *streams* replica tudo.

De acordo com PostgreSQL (2010b), o recurso de SR (*Streaming Replication*) do PostgreSQL envia os dados entre os servidores no formato de registros WAL (*Write-Ahead Logging*) XLOG. Ainda segundo PostgreSQL (2010b), o WAL é um método padrão para assegurar a integridade de dados.

3.1.3 Mamooth

O Mamooth[4] é uma versão modificada do PostgreSQL, que adiciona suporte a replicação de dados em modo assíncrono. Utiliza-se de *logs* de transações distribuídas para gerenciar as atualizações geradas por uma base de dados mestre. A documentação para esta ferramenta é bastante escassa, o que dificulta o estudo a respeito desta.

De acordo com a documentação do Mamooth[5], algumas das características dessa ferramenta são: Replicação *master* para vários *slaves;* suporte para replicação de *large objects*; replicação em lotes (utilizado para replicação através de conexões periódicas, ou seja, que não está disponível o tempo todo); não possui suporte para replicação das réplicas em cascata; e não possui suporte a replicação de DDL.

3.2 Replicação de dados no Oracle

O banco de dados Oracle possui diversos recursos nativos que possibilitam replicação entre bases de dados. Para isto torna-se necessário a utilização de *database links*.

---

[4] Disponível em: <http://www.commandprompt.com/products/mammothreplicator> Acesso em: 10 out. 2010

[5] Disponível em: <http://wiki.postgresql.org/wiki/Mammoth> Acesso em: 22 jun. 2011.

Este é um recurso do banco de dados Oracle que possibilita estabelecer uma conexão entre diferentes bases de dados. Com isto é possível executar comandos DML como inserção, atualização, exclusão em uma base de dados externa.

Recursos como visões materializadas, *streams*, ou até mesmo processamento em lote podem ser facilmente utilizados para proporcionar replicação entre bases de dados quando utilizados em conjunto com *database links*.

3.2.1 Replicação com visões materializadas

De acordo com Oracle (2003), uma visão materializada é uma cópia integral ou parcial de uma ou mais tabelas de um banco de dados a partir de um único ponto no tempo e que tem sua estrutura baseada em um comando de seleção. Diferente das visões comuns, as visões materializadas possibilitam a interação com o usuário, mesmo enquanto este estiver desconectado do servidor de banco de dados central. Sendo possível efetuar uma sincronização da visão, quando a conexão com o banco de dados central estiver disponível, atualizando-a com todas as alterações que possam ter acontecido enquanto a conexão estava indisponível.

Segundo Oracle (2010a), as visões materializadas podem ser classificadas como:

- De leitura: esse tipo de visão não pode ser modificada, seus dados só podem ser atualizados executando uma sincronização com a tabela.
- Atualizável: permite aos usuários executar qualquer comando DML, sendo que neste caso a tabela de origem será sincronizada com a visão, ou seja, o comando executado sobre a visão terá efeito também sobre a tabela de origem. No entanto, este tipo de visão está restrito a ter apenas uma única tabela como base.
- De escrita: permite que sejam efetuadas alterações em seus dados, mantendo a tabela de origem intacta. Porém, todas as alterações são perdidas quando esta for novamente sincronizada.

A sincronização de uma visão materializada é fundamental para garantir que os dados estejam consistentes com as tabelas do banco de dados. E pode ser realizada de duas formas: completa, onde a visão é completamente atualizada, ou rápida, onde apenas os dados que foram modificados desde a última sincronização são atualizados. Para que seja possível efetuar uma sincronização rápida, é necessária a utilização de um *log* de visão materializada. Esse *log* contém todas as alterações DML efetuadas na tabela do banco de dados. Ela é

associada a apenas uma tabela. Quando é realizada uma sincronização rápida sobre a visão materializada, esse *log* é lido para verificar todas as alterações ocorridas desde a última sincronização, e então aplicá-las para a visão.

As visões materializadas são um recurso nativo do banco de dados Oracle. No entanto, podem ser executadas com base em tabelas de um banco de dados distinto. Para isso é necessário a utilização de outra ferramenta da Oracle, chamada *Oracle Transparent Gateway*, que permite uma comunicação transparente entre o SGBD Oracle e um SGBD distinto.

3.2.2 Replicação com *procedures*

Aplicações de processamento em lote muitas vezes pode ser a maneira mais simples para replicação de estruturas complexas. No entanto, essas rotinas podem sobrecarregar uma rede com um intenso tráfego gerado quando modificam um grande volume de dados dentro de uma única transação. De acordo com Oracle (2003), a maneira mais comum de utilizar esse método de replicação é criar um procedimento que verifique o objeto que deve ser atualizado, e efetue atualização do *site* destino. Outra maneira mais otimizada para este tipo de replicação é manter as estruturas de dados idênticas em todos os *sites*, e também o procedimento que efetua a alteração dos dados, sendo possível assim executar uma chamada remota ao procedimento para que este efetue as alterações localmente em cada base de dados, diminuindo assim o tráfego de dados na rede.

A utilização de *procedures* também pode proporcionar ao Oracle, replicação entre bases de dados distintas. O banco de dados permite compilar pacotes e classes de código Java, e utilizá-los de maneira transparente, como se fosse um objeto PL/SQL. Dessa forma, é possível compilar um pacote de classes JDBC (*Java Database Connectivity*) para proporcionar a comunicação com um banco de dados não Oracle, habilitando o desenvolvedor a construir um procedimento para interagir com essa base de dados naturalmente.

3.2.3 *Oracle streams*

Oracle streams é um dos recursos para integração e replicação de dados no SGBD Oracle. Ele pode ser aplicado em um ambiente homogêneo, tanto quanto em um ambiente heterogêneo com a utilização do *Oracle Transparent Gateway*. Oracle *streams* provê um recurso para propagação de alterações ocorridas no banco de dados. De acordo com Oracle

(2002), a replicação com o recurso de *streams* consiste em basicamente três etapas: primeiro as alterações são formatadas como LCR (*Logical Change Record*) e então armazenadas em filas, em seguida as alterações são propagadas para uma nova fila já na outra base de dados, e por último a alteração é aplicada ao objeto de destino.

Ainda segundo Oracle (2002), este método de replicação também permite a replicação entre esquemas diferentes. É possível efetuar regras de transformações nas etapas de captura ou propagação. Em caso de replicação entre um SGBD heterogêneo, o Oracle pode aplicar as alterações diretamente no banco de dados destino, sem propagar os LCRs para a base destino.

A captura de alterações no banco de dados consiste em extrair mudanças ocorridas em tabelas, esquemas, ou em todo o banco de dados utilizando os arquivos de *redo log* ou *archived log*. Um arquivo de *redo log* grava todas as alterações efetuadas em um objeto, quando este é submetido a uma operação de *commit* ou *rollback*. As alterações podem ser tanto de DML como de DDL.

A grande desvantagem da utilização deste recurso é a necessidade da aquisição de outro produto (*Transparent Gateway*), e a grande necessidade de desenvolvimento, visto que o *Oracle Streams* é disponibilizado no banco de dados através de rotinas como pacotes e procedimentos.

3.2.4 *Oracle Transparent Gateway*

O *Oracle Transparent Gateway* permite integrar dados entre bases de dados heterogêneas. A utilização deste recurso permite acessar bases de dados não Oracle de forma transparente aos usuários, dando a impressão de que todos os objetos pertencem a um único banco de dados Oracle. Além disso, o *Oracle Transparent Gateway* garante também a transparência nos comandos SQL, sendo desnecessário ao usuário preocupar-se com o dialeto SQL do banco de dados heterogêneo, podendo utilizar qualquer recurso PL/SQL com qualquer objeto do banco de dados.

Com esta ferramenta, recursos como *procedures* e visões materializadas podem ser usadas no desenvolvimento de rotinas de replicação entre bases de dados distintas. No entanto, de acordo com a documentação do *Oracle Transparent Gateway*, uma limitação deste recurso é que o SGBD não Oracle deve ser o servidor *master*.

3.2.5 *Oracle GoldenGate*

O *Oracle GoldenGate* é uma ferramenta da Oracle para replicação de dados, e que deve ser adquirida separadamente do banco de dados. Esta ferramenta utiliza-se de arquivos de *redo logs* para capturar as mudanças ocorridas em uma base de dados. No entanto, a captura dessas mudanças é realizada por uma camada de *software* externa ao banco de dados. A sincronização consiste em três etapas: etapa de captura, etapa de distribuição e etapa de gerenciamento.

De acordo com Oracle (2010c), a etapa de captura, consiste em capturar comandos DML (*insert*, *update*, *delete*) do banco de dados origem e organizá-los em filas, que geralmente são gravadas em arquivos binários, externos ao banco de dados. A etapa de distribuição é responsável por ler a fila gravada na etapa de captura e aplicar os comandos na base de dados destino. O módulo de gerenciamento consiste em gerenciar todo o processo de replicação.

Tumma (2005) afirma que o *GoldenGate* é um sistema multiprocessado que não é afetado pelos níveis de recurso do banco de dados. Como seus processos são executados externamente ao banco de dados, ele não causa nenhum *overhead* no servidor primário do banco de dados.

3.2.6 *Oracle Data Integrator*

O *Oracle Data Integrator* é uma plataforma de desenvolvimento de módulos para replicação e integração de dados. O ODI era conhecido anteriormente como *Sunopsis*, uma ferramenta desenvolvida na França que foi adquirida pela Oracle em meados de 2006. Da mesma forma como o *GoldenGate*, deve ser adquirido separadamente do banco de dados.

Ela também oferece suporte a cargas e transformações em grandes volumes de dados. O ODI provê três modelos de arquitetura para integração de dados: baseado em eventos, serviços ou dados. A integração baseada em eventos utiliza um recurso chamado *Changed Data Capture* (CDC) para captura das modificações na base de dados; a integração baseada em dados é utilizada para o processamento em lote de grandes massas de dados; e a integração baseada em serviços utiliza-se da suíte SOA (*Service Oriented Architecture*) da Oracle.

De acordo com Oracle (2010d), o recurso de CDC possibilita ao ODI registrar alterações causadas por outras aplicações na origem dos dados. O funcionamento deste recurso é baseado em um modelo documentação (ou *journalizing*).

O ODI conta com dois modelos de documentação: o *Simple Journalizing* e o *Consistent Set Journalizing*. O primeiro registra alterações em um *datastore* individual. Já o *Consistent Set Journalizing* registra alterações em um grupo de *datastores* validando a integridade referencial entre os mesmos.

O ODI pode integrar dados entre bases heterogêneas, o que é essencial para aplicação da ferramenta no contexto da empresa desse estudo. Sua arquitetura é baseada no conceito de E-LT (*Extract Load Transform*). Este modelo de arquitetura é similar ao ETL. No entanto, em um modelo ETL tradicional, existe uma unidade central de transformação, já no E-LT, primeiramente é efetuada a carga dos dados para o destino, para depois, utilizando recursos nativos do SGBD, efetuar a transformação dos dados.

O modelo E-LT do *Oracle Data Integrator* proporciona maior desempenho no processo de transformação comparado a um processo ETL convencional. Todo o desenvolvimento é otimizado para o banco de dados em questão com o uso de funções nativas do próprio, como será demonstrado posteriormente. Existem módulos a serem utilizados com bancos de dados, que implementam a linguagem nativa do mesmo. Assim sendo, a linguagem utilizada em cada banco de dados do processo será nativa, mesmo que os SGBD sejam distintos.

O desenvolvimento de novos módulos de integração E-LT são realizados em modo declarativo, ou seja, o desenvolvedor não necessita programar todo o processo de como a integração deve ser realizada, é preciso apenas declarar o que precisa ser feito, enquanto o ODI encarrega-se do processo de integração. As regras de negócio do ODI definem qual o destino de um dado específico, e qual regra de transformação deve ser utilizada.

Para controle de acesso o ODI, pode mapear usuários definidos no LDAP (*Lightweight Directory Access Protocol*), Oracle *Internet Directory* ou *Active Directory*. Com isso a autenticação pode ser delegada ao sistema de autenticação da corporação, habilitando um serviço SSO (*Sign-On)* do ODI, mantendo as configurações de usuário e senha em um local centralizado, garantindo os padrões de segurança da empresa.

O *Oracle Data Integrator* também é capaz de gerenciar versões de unidades de trabalho, que são objetos desenvolvidos na ferramenta, como *interfaces*, pacotes, procedimentos, etc. Ele é capaz de proteger e restaurar versões de projetos, modelos de interface e assim por diante, dispensando assim, a utilização de uma ferramenta exclusiva para controle de versões. Em um ambiente com múltiplos repositórios, uma unidade de trabalho pode ser encontrada em diversos locais por existir a necessidade de replicar processos. Este cenário apresenta o risco de uma unidade de trabalho mais recente ser sobreposta por uma

versão anterior. Porém o ODI elimina este risco armazenando todas as versões de unidades no repositório mestre.

O ODI foi desenvolvido baseado em um sistema de repositórios que armazenam um conjunto de metadados referentes a cada projeto. O conjunto de repositórios do ODI fica armazenado em um SGBD, e é formado obrigatoriamente por um repositório mestre e no mínimo um repositório de trabalho. Enquanto o repositório mestre contém informações sobre versões de objeto, topologia e segurança, objetos de projeto são armazenados nos repositórios de trabalho. Este pode conter modelos, projetos, e informações de *runtime*. Informações de *runtime*, ou tempo de execução consistem em informações sobre agendamento de tarefas, *logs*, e outros. Com base no conteúdo dos repositórios, o ODI pode ainda, gerar relatórios com a documentação sobre todo o processo de integração.

Cada unidade de trabalho do ODI geralmente está relacionada, ou é dependente de algum outro elemento ou objeto. O repositório de metadados do ODI mantém todas as referências entre objetos. De acordo com Oracle (2010b), metadados são frequentemente descritos como "a informação sobre os dados". Mais precisamente, os metadados são a descrição dos dados em si; sua finalidade; como são usados; e os sistemas usados para gerenciá-los. Isto normalmente inclui definições de modelos, transformações, informações sobre fluxos de processos, *logs*, descrições sobre sistemas de BI (*Business Intelligence*), e qualquer outra informação relevante.

A padronização dos processos de ETL é um dos grandes benefícios providos pela utilização desta ferramenta. Por seu desenvolvimento ser baseado em fluxogramas, e não em linhas de código, a ferramenta proporciona ainda grande redução de trabalho na construção de módulos para integração de dados, gerando interfaces de fácil compreensão e manutenção, eliminando complexas estruturas de códigos.

Para auxiliar a definição de modelos de estruturas, o ODI possui recursos de engenharia reversa. Isso possibilita a obtenção do DDL da estrutura original diretamente da origem, gerando um modelo de dados fiel à estrutura da base de dados. Dessa forma, é necessário apenas indicar o que deseja ser replicado.

De acordo com Oracle (2010b), um modelo de dados é um conjunto de *datastores* correspondentes a uma estrutura de dados de um esquema físico. Esses modelos podem ser organizados em pastas, e podem ser criados utilizando engenharia reversa. O *datastore* descreve os dados no formato de estrutura de uma tabela ou visão. Sendo basicamente composto de colunas, pode também conter elementos como chaves, referências restrições ou filtros.

O ODI é composto por um conjunto de módulos que cumprem finalidades específicas. Segundo Oracle (2009), integra esse conjunto o *Topology Manager*, *Designer*, *Security Manager* e o *Operator*.

- *Topology Manager*: utilizado para gerenciar a topologia, ou seja, a arquitetura física e lógica do ambiente. Possui uma série de arquiteturas pré-definidas, que são objetos que representam servidores de dados, arquivos ou até mesmo sistemas operacionais;
- *Designer*: utilizado no desenvolvimento de projetos e manutenção de *interfaces* de integração. Neste são definidas as sentenças declarativas para transformação e integração dos dados. Os módulos desenvolvidos pelo *designer* são armazenados no repositório de trabalho. O desenvolvimento destes baseia-se essencialmente na construção de diagramas onde são definidos o fluxo do processo, a origem e destino da informação;
- *Security Manager*: é utilizado para gerenciar usuários e permissões de acesso. Neste módulo é possível definir perfis de usuários, bem como permissões de acesso a objetos ou *hosts*;
- *Operator*: utilizado para monitoração e gerenciamento de sessões e execuções. Através dele é possível visualizar o código do um programa em execução, bem como estatísticas relacionadas ao mesmo. Como quantidade de registros processados, número de inserções, alterações, exclusões, e tempo de execução. Pode-se inclusive interagir com um processo em execução efetuando o encerramento ou reinício do mesmo. É possível também visualizar não apenas os processos em execução, como também informações de processos históricos, funcionando também como um gerenciador de *log* de execuções.

3.3 A ferramenta escolhida

Para o desenvolvimento dos experimentos foi escolhida a ferramenta *Oracle Data Integrator*, em função de esta possuir a capacidade de realizar a replicação de dados entre SGBD distintos, sem a necessidade de aquisição de outra ferramenta adicional para cumprir este propósito. Os experimentos são realizados com o intuito de comprovar a eficácia da ferramenta diante dos problemas apresentados no contexto de replicação de dados na empresa. Além da descrição dos experimentos no capítulo final, são apresentadas no próximo capítulo, as etapas de criação de um processo de replicação de dados com o ODI.

4 REPLICAÇÃO DE DADOS COM ODI

O desenvolvimento de rotinas de replicação de dados no *Oracle Data Integrator* é realizado em modo declarativo. O desenvolvimento é baseado em módulos, chamados pela ferramenta de *interface*. A seguir é descrito o processo de criação de uma *interface* de replicação.

A primeira etapa para criação de um módulo de replicação é a definição das topologias através do *Topology Manager*. Por definição de topologias compreende-se a criação de arquitetura física, lógica e o contexto da aplicação.

A arquitetura física representa a conexão física com o servidor de dados. Para a realização dos testes são necessárias duas arquiteturas físicas: uma de tecnologia Oracle, representando o servidor central, outra PostgreSQL, representando a base de dados das filiais. O ODI possui diversas tecnologias pré-definidas bastando apenas configurar o acesso à arquitetura referente à tecnologia. Existe a possibilidade de definir-se uma tecnologia completamente nova, no entanto isto está fora de escopo e não será abordado neste trabalho.

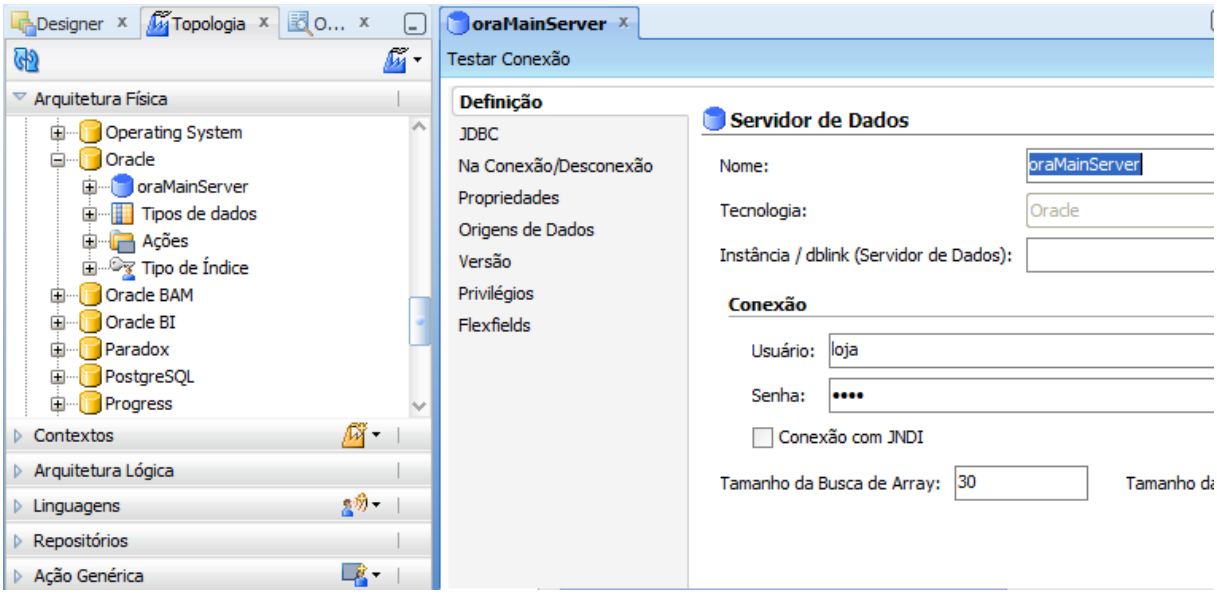

Figura 4.1 - Definição de arquitetura física

A Figura 4.1 exibe a topologia do servidor central. Para definir o acesso físico a essa base de dados é necessário informar o usuário e senha, e as informações de *host* e porta na aba de configurações do JDBC, bem como o *driver* JDBC utilizado para realizar a conexão com este SGBD. Essa configuração deve ser feita para cada uma das bases de dados que estarão envolvidas com a replicação. No caso do PostgreSQL o *driver* JDBC não é disponibilizado com instalação do *Oracle Data Integrator*, sendo necessário efetuar a instalação manualmente

do mesmo. Para efetuar a instalação manualmente basta disponibilizar o arquivo na pasta "userlib" do diretório do ODI.

Tendo definidas todas as arquiteturas físicas é necessário definir uma arquitetura lógica. Este é um objeto que representa logicamente a conexão física. Durante o desenvolvimento do projeto a conexão é referenciada pela arquitetura lógica e não pela física. A ligação entre uma arquitetura lógica e física é realizada através de um contexto. Um contexto define onde um projeto do ODI será executado. Pode-se utilizar um contexto para o ambiente de produção e outro para o ambiente de homologação, sem necessidade de efetuar alterações no projeto.

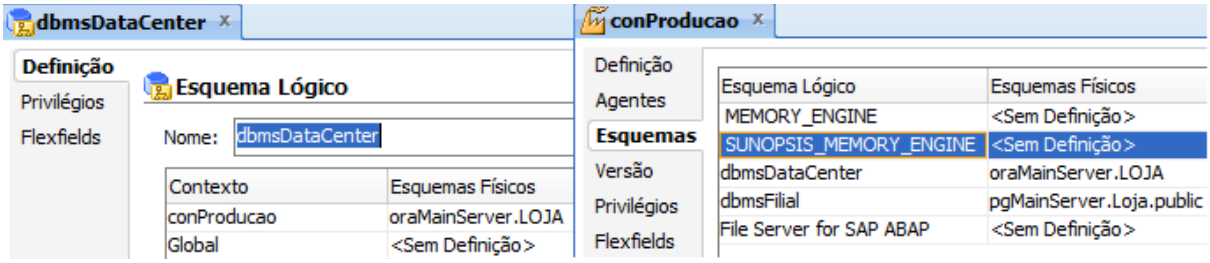

Figura 4.2 - Esquema lógico e contexto

A Figura 4.2 exibe as principais propriedades do esquema lógico "dbmsDataCenter", e do contexto "conProducao". Para a criação de um esquema lógico basta definir o seu nome, pois o seu relacionamento com o esquema físico é definido no contexto. Nesta situação o esquema lógico está direcionado ao esquema "LOJA" no servidor "oraMainServer", enquanto o esquema "dbmsFilial" aponta para o esquema "public" do banco de dados "Loja" no servidor "pgMainServer". Finalizadas essas configurações é possível iniciar o desenvolvimento do projeto através do módulo *Designer*.

A primeira etapa para o desenvolvimento do projeto é a definição dos módulos de conhecimento que serão utilizados. Os *Knowledge Modules* (KM) possuem o conhecimento para executar um conjunto específico de tarefas. Desses conjuntos de tarefas destacam-se carga, validação e integração. Os KMs devem ser importados a cada novo projeto. É importante importar apenas os módulos necessários para o desenvolvimento do projeto em questão.

De acordo com Oracle (2010), os módulos de conhecimento são organizados em tipos, de acordo com o conjunto de atividades às quais possuem conhecimento:

- CKM *Check Knowledge Module*: responsável por verificar a consistência dos dados. Pode realizar validações como obrigatoriedade de campos, condições, e *constraints* do banco de dados, ou até mesmo definidas no próprio ODI. Cada

CKM pode conter diversas verificações, sendo possível ao desenvolvedor selecionar quais devem ser realmente executadas;

- IKM *Integration Knowledge Module*: Responsável pela integração dos dados na base de dados destino, de acordo com as regras de ETL definidas nas interfaces. Os dados podem ser integrados em modo *update*, onde novos registros são inseridos (*insert*) e registros existentes são atualizados (*update*), ou em modo *replace/append*, onde a tabela destino é truncada e todos os registros são atualizados através de uma inserção (*insert*);
- JKM *Journalizing Knowledge Module*: módulos utilizados para trabalhar com *journalizing*, e recursos *Changed Data Capture* (CDC);
- LKM *Load Knowledge Module:* responsável pela carga de dados a partir da origem;
- RKM *Reverse Knowledge Module:* pode ser utilizado para realizar rotinas customizadas para engenharia reversa e obtenção de metadados;
- SKM *Service Knowledge Module*: pode ser utilizados para manipular dados via *Web Services.*

Cada tecnologia pode ter os seus KMs específicos. Por exemplo, existem KMs específicos para Oracle, DB2, Microsoft SQL Server. Entretanto também existe um módulo genérico para trabalhar com bases de dados SQL. Como não existem módulos específicos para o PostgreSQL foram utilizados os módulos genéricos de SQL para trabalhar com este SGBD. Se necessário fosse, poderiam ser desenvolvidos módulos específicos para o PostgreSQL, pois a ferramenta possui essa flexibilidade. No entanto, o desenvolvimento de módulos de conhecimento não está no escopo desse trabalho.

Para o banco de dados Oracle, ao contrário do PostgreSQL o ODI dispõe de diversos módulos específicos, a maior variedade está nos módulos de integração e *journalizing* de dados. Cada tipo de módulo possui várias versões que cumprem finalidades específicas. Abaixo está listada uma breve descrição do comportamento de cada módulo específico para o SGBD Oracle, segundo Oracle (2010b).

- CKM Oracle: Verifica a integridade de *constraints* definidas na tabela do banco de dados.
- IKM Oracle *Incremental Update*: A integração dos dados é efetuada utilizando os comandos DML *insert* e *update*.

- IKM Oracle *Incremental Update (Merge)*: A atualização dos dados é efetuada utilizando os comandos DML *merge*.
- IKM Oracle *Incremental Update* (PL/SQL): A integração é realizada através de recursos da linguagem PL/SQL.
- IKM *Oracle Multi Table Insert*: Insere dados em múltiplas tabelas através do comando de *multi-table insert* (MTI).
- IKM Oracle *Slowly Changing Dimension*: Integra dados em uma dimensão de um DW, baseada no SCD tipo 2. De acordo com *Oracle® Fusion Middleware Developer's Guide for Oracle Data Integrator 11g*, em uma ambiente SCD tipo 2 existem múltiplas versões de um mesmo registro na dimensão, e novas versões são criadas, á medida que registros antigos permanecem inalterados.
- IKM Oracle *Spatial Incremental Update*: Utiliza o comando *merge* para efetuar a atualização dos dados e provê suporte aos tipos de dados SDO_GEOMETRY.
- IKM Oracle to Oracle *Control Append* (DBLINK): Integram dados através de diferentes instâncias do banco de dados Oracle.
- IKM Oracle AW *Incremental Update*: Além de realizar a integração de dados este módulo possibilita efetuar a atualização de um cubo em um ambiente OLAP.
- JKM Oracle *10g Consistent (Streams)*: Cria a estrutura de *journalizing* em uma tabela Oracle utilizando os recursos de *Streams* do Oracle 10g.
- JKM Oracle *11g Consistent (Streams)*: Cria a estrutura de *journalizing* em uma tabela Oracle utilizando os recursos de *Streams* do Oracle 11g.
- JKM Oracle *Consistent*: Cria a estrutura de *journalizing* em uma tabela Oracle utilizando *triggers*.
- JKM Oracle *Consistent (Update Date)*: Cria a estrutura de *journalizing* em uma tabela Oracle utilizando *triggers* baseadas em um campo do tipo *date* contendo a data da última atualização.
- JKM Oracle *Simple*: Cria a estrutura de *journalizing* em uma tabela Oracle utilizando *triggers*. Este módulo está sujeito a violações de chave, pois ao contrário dos módulos *consistent*, este não garante a integridade das chaves de referência.

- JKM Oracle *to* Oracle *Consistent* (OGG): Cria a estrutura de *journalizing* em uma tabela Oracle quando é utilizado o *Oracle GoldenGate*.
- LKM Oracle BI *to* Oracle (DBLINK): Carrega dados de uma camada física do Oracle BI para uma tabela Oracle.
- LKM Oracle *to* Oracle (DBLINK): Carregam dados através de instâncias do Oracle utilizando DBLINK.
- LKM Oracle *to* Oracle (*datapump*): Carrega dados entre instâncias do Oracle utilizando o recurso de *external table*. Este recurso possibilita acessar dados externos como se fosse uma tabela nativa do Oracle.
- LKM SQL *to* Oracle: Carrega dados de uma base de dados compatível com o SQL-92 ANSI.

Para o SGBD PostgreSQL não existem módulos de conhecimento específicos, sendo necessário utilizar os módulos compatíveis com ANSI SQL. Para ANSI SQL não existe uma variedade tão grande de módulos como para Oracle. No entanto os módulos IKM, LKM, CKM podem ser utilizados perfeitamente para realizar a replicação de dados com o SGBD PostgreSQL. Ainda conforme Oracle (2010b) segue abaixo a descrição dos principais módulos compatíveis com ANSI SQL-92.

- CKM SQL: Verifica a integridade de *constraints* definidas na tabela de um banco de dados ANSI SQL-92.
- IKM SQL *Control Append*: Integra dados à base de dados destino truncando dados existentes, modo *replace/append*.
- IKM SQL *Incremental Update*: A integração dos dados é efetuada utilizando os comandos DML *insert* e *update*.
- IKM SQL *to* SQL *Control Append*: Similar ao módulo SQL *Control Append*, no entanto é utilizado para diferentes SGBD compatíveis com ANSI SQL-92.
- IKM SQL *to* SQL *Incremental Update*: Similar ao SQL *Incremental Update*, porém deve ser utilizado para diferentes SGBD compatíveis com ANSI SQL-92.
- LKM SQL *to* SQL: Carrega dados entre diferentes SGBD compatíveis com ANSI SQL-92.
- LKM SQL *to* SQL (*row by row*): Carrega dados entre SGBD compatíveis com a ISO-92 utilizando *scripts* da linguagem Jython.

- LKM SQL *to* SQL (JYTHON): Carrega dados entre diferentes SGBD compatíveis com a ANSI SQL-92 utilizando *scripts* da linguagem Jython.

Dando continuidade ao desenvolvimento do projeto, a Figura 4.3 apresenta os módulos de conhecimento que foram carregados para o desenvolvimento do projeto. Durante a etapa de testes serão verificados diversos módulos. Sendo assim o nome dos módulos utilizados serão apresentados junto aos resultados obtidos.

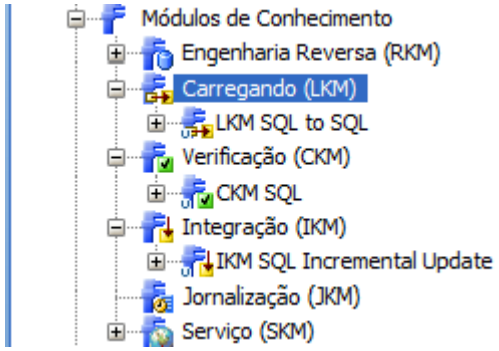

Figura 4.3 - Módulos de conhecimento carregados

Definidos e adicionados ao projeto os módulos de conhecimento, é necessário definir os modelos de dados que serão utilizados. Conforme comentado anteriormente, um modelo de dados é um conjunto de *datastores* correspondentes a uma estrutura de dados de um esquema físico Estes podem ser definidos utilizando-se recursos de engenharia reversa do ODI. A Figura 4.4 exibe a estrutura de alguns modelos de dados definidos, mesmo sendo organizados em pastas, não podem existir dois modelos com o mesmo nome, por isso foi adicionado o sufixo "_ORA" ou "_PG" para distinguir entre os modelos da base Oracle ou PostgreSQL.

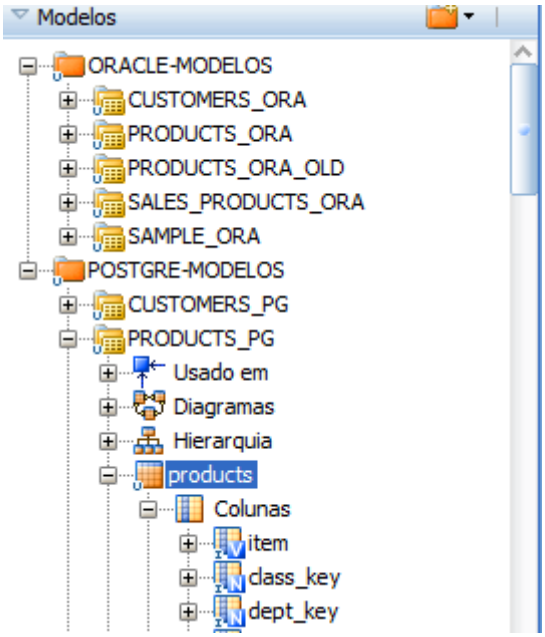

Figura 4.4 - Modelos de dados definidos

Com os modelos de dados definidos deve-se criar a *interface* responsável pela replicação dos dados. Uma *interface* é um objeto que mantém um conjunto de regras que definem a carga de um servidor de dados de destino. É na *interface* que são definidas a origem, destino e mapeamento de dados. A Figura 4.5 apresenta o mapeamento entre uma tabela do Oracle "PRODUCTS" e uma tabela do PostgreSQL "products". Esse mapeamento é feito de forma automática quando são adicionadas as estruturas de origem e destino à *interface*, se estas possuírem os mesmos nomes de colunas em sua estrutura. Caso contrário é necessário definir o mapeamento manualmente entre as colunas de origem e destino.

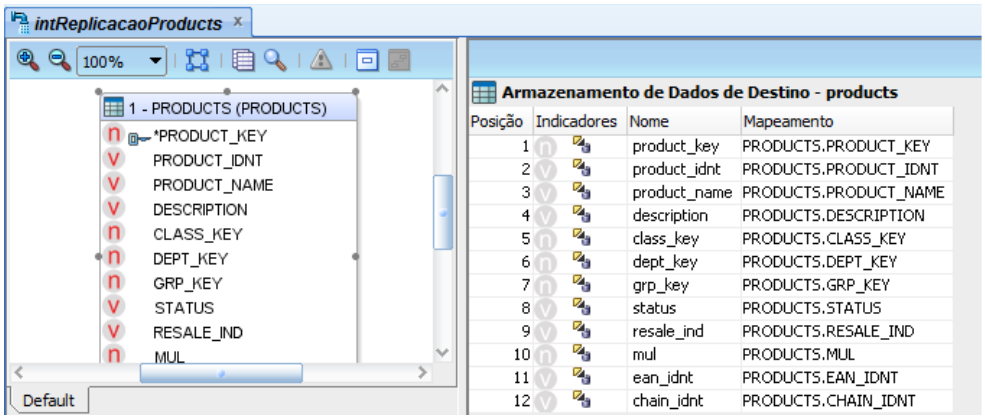

Figura 4.5 - Mapeamento de dados

Na *interface* são definidos também informações como origens, que podem ser mais de uma, no caso de uma seleção (*select*). É possível determinar um contexto de execução distinto para cada uma das estruturas de origens. Também podem ser definidos filtros restringindo dados das tabelas de origem. A Figura 4.6 apresenta a definição de um filtro. É possível definir qualquer expressão SQL como filtro, e ainda definir se deve ser executado na origem ou destino.

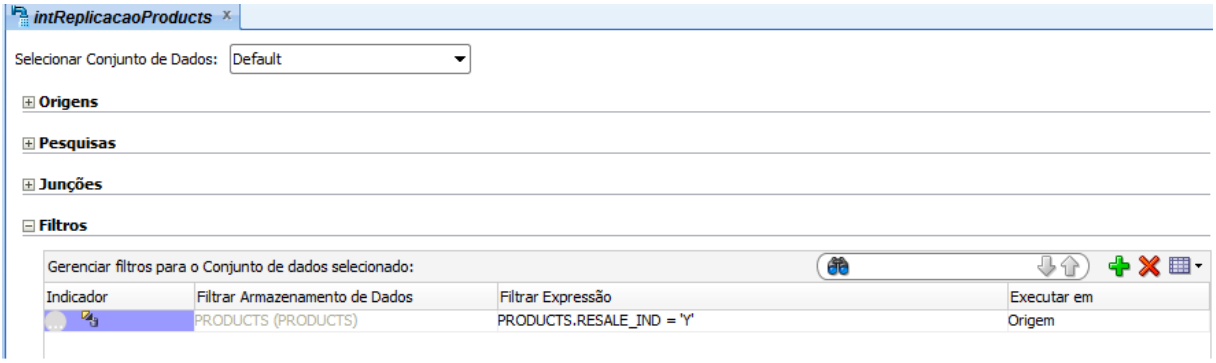

Figura 4.6 - Definição de filtros na interface

Após terem sido definidos os mapeamentos, é possível visualizar o diagrama de execução da *interface*, conforme apresentado na Figura 4.7. Com esse diagrama torna-se clara a compreensão do funcionamento do módulo. O ODI carrega os dados da tabela

"PRODUCTS" do servidor "oraMainServer" para uma tabela de *stage* no servidor "pgMainServer" onde serão efetuados os testes pelo módulo de validação (CKM) e em seguida os dados validados serão integrados à tabela destino "products" do mesmo servidor.

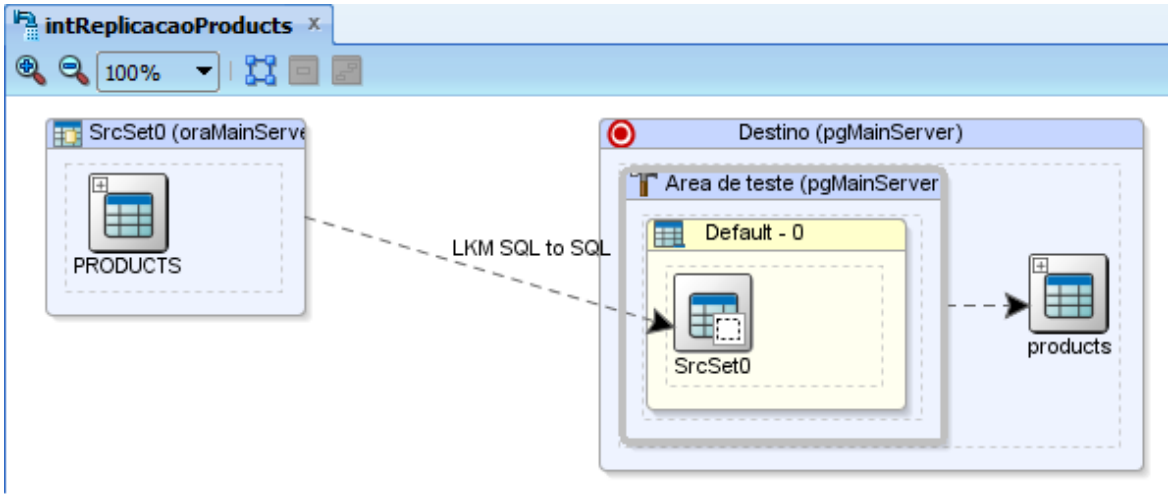

Figura 4.7 - Fluxo de execução

A Figura 4.7 representa o fluxo de execução para uma replicação total da tabela. Ou seja, todos os dados da tabela de origem serão replicados ao destino. O *Oracle Data Integrator* possibilita que sejam realizados junções e filtros em tabelas da base de dados de destino. Como mostra a Figura 4.8, nessa imagem foi realizada uma junção da tabela FISCAL_HEADER (que corresponde aos dados da nota fiscal) do banco de central, com a tabela COMPANYES (da base de dados destino).

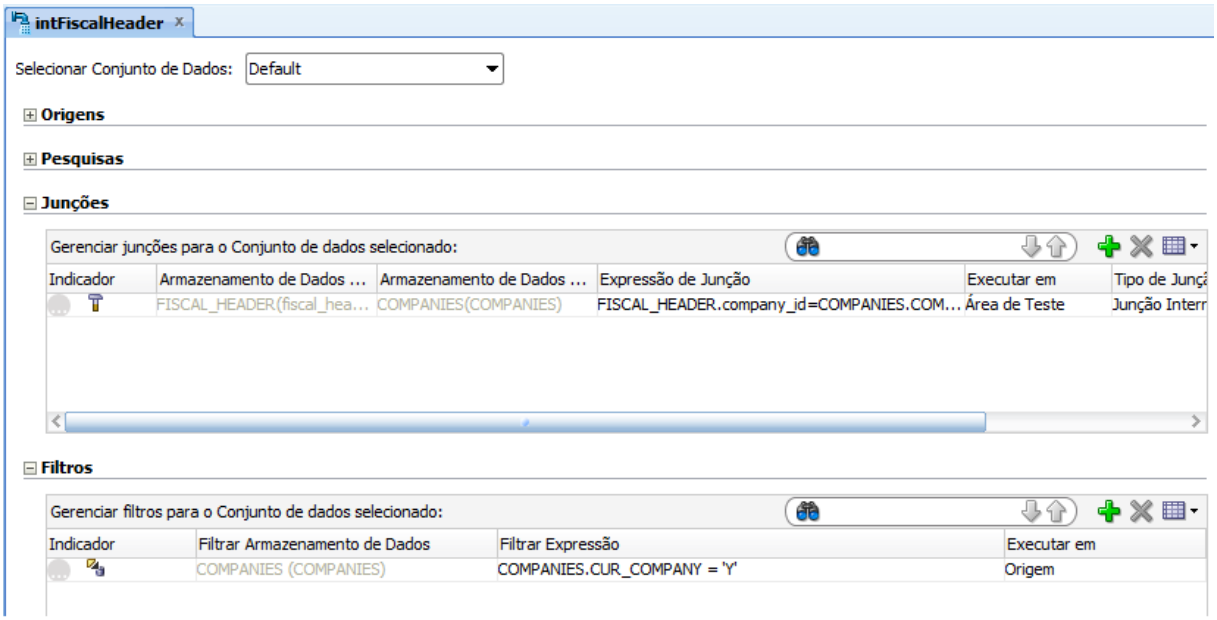

Figura 4.8 - Junção e filtro na base de dados destino

O fluxo de execução do processo acima é mais bem descrito pela Figura 4.9, onde os dados da nota fiscal (FISCAL_HEADER) são replicados para a área de testes no banco de dados de destino, onde é realizada uma junção e um filtro com a tabela de filiais (COMPANIES). Isto é feito para que os dados a serem integrados à tabela de notas fiscais da base da unidade sejam unicamente informações referentes à unidade em questão.

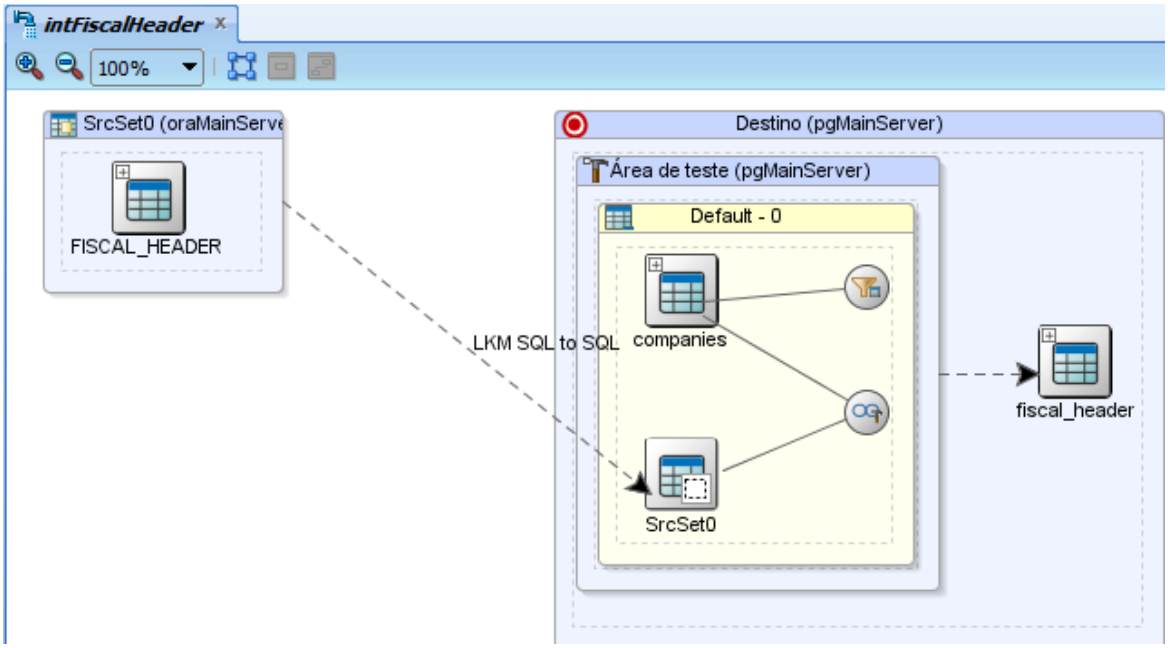

Figura 4.9 - Fluxo de execução da junção com tabela no destino

No desenvolvimento de *interfaces* podem ser utilizados ainda objetos como procedimentos, funções e variáveis, assim como em qualquer linguagem de programação. O ODI possibilita que procedimentos e funções sejam criados em qualquer linguagem das tecnologias definidas no *Topology Manager*. Para o desenvolvimento dessas rotinas pode-se inclusive utilizar a API do *Oracle Data Integrator*.

Sequências também podem ser definidas pelo ODI. Elas apresentam o mesmo comportamento que uma sequência de bancos de dados. Uma sequência definida pelo ODI pode mapear uma sequência nativa do banco de dados. Esta pode ser utilizada mesmo quando o SGBD não apresentar este recurso. Nesses casos o ODI é quem controlará o incremento da sequência. Para os modelos replicados nos testes deste trabalho não foi necessário criar procedimentos, funções ou mesmo sequências, visto que apesar das bases de dados serem distintas, as estruturas das tabelas são homogêneas.

Uma *interface* é passível de execução. Ou seja, neste ponto já é possível executar a *interface* e acompanhar o processo de carga da tabela de destino. No entanto, a execução da *interface* só pode ser feita manualmente sendo que para dar continuidade ao projeto, e para realizar o *deploy* do módulo é necessário gerar um cenário da *interface* desenvolvida.

O cenário possibilita que sejam feitos agendamentos para execução do módulo. Um cenário é uma espécie de versão compilada de um objeto, e devem ser gerados para efetuar o *deploy* de objetos como *interfaces*, procedimentos e pacotes. A Figura 4.10 exibe o cenário do módulo de integração de clientes sendo configurado para agendamento. Os agendamentos de cenários são gerenciados por um agente. Um agente é um serviço que funciona como um *listener* (ouvinte) em uma porta TCP/IP.

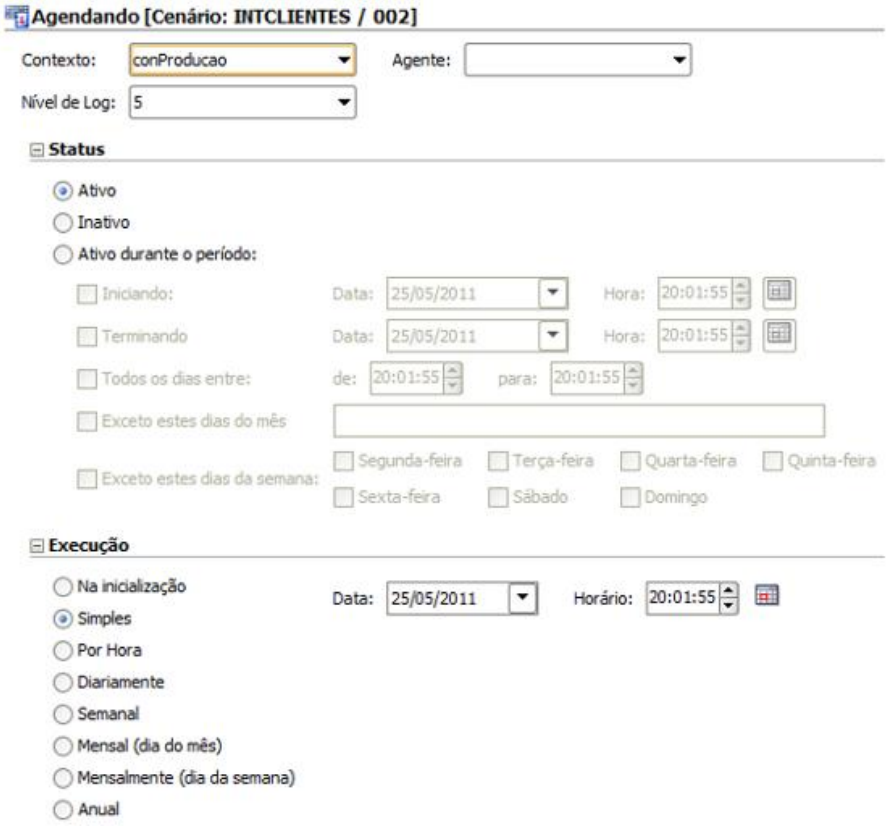

Figura 4.10 - Agendamento de execução do cenário

Um pacote é um objeto cuja finalidade é empacotar outros objetos como *interfaces* e procedimentos e outros cenários. Um pacote por sua vez também gera um cenário. No pacote pode ser desenvolvido um diagrama complexo, com uma variedade maior de recursos do que apenas na *interface*. No pacote, bem como em procedimentos e módulos de conhecimento, é possível desenvolver melhores rotinas de controle utilizando as ferramentas *Open Tools*.

De acordo com Oracle (2009), as ferramentas *Open Tools* são comandos que podem ser usados em tempo de execução, e possibilitam desenvolver fluxos alternativos capacitando a desenvolver rotinas específicas de controle. Além disso, possuem recursos para manipular arquivos externos a um SGBD, interagir com o próprio sistema operacional, ou até mesmo trabalhar com a internet, detecção de eventos, entre outros.

Além dos comandos pré-instalados, é possível adicionar novas classes as *Open Tools*. Pode-se também desenvolver ferramentas personalizadas. Como estas são classes desenvolvidas na linguagem Java é possível estendê-las criando novos componentes para utilizá-las no ODI.

No *Oracle Data Integrator 11g* as *Open Tools* estão divididas nos seguintes grupos:

- Arquivos: Permite trabalhar com arquivos, possibilita realizar atividades como copiar, mover, excluir compactar.
- Captura de dados alterados: Comandos de *journalizing*, que permitem obter dados alterados por outras aplicações.
- Detecção de eventos: Pode detectar eventos no sistema operacional, como um arquivo que foi criado em determinado diretório, ou eventos na base de dados como, por exemplo, a inserção da quantidade X de registros em uma tabela Y.
- *Internet*: Comandos para trabalhar com SCP (*Secure Copy*) ou FTP (*File Transfer Protocol*), ou realizar chamadas a um *WebService*.
- Metadados: Recursos para obter metadados de estruturas de um SGBD.
- Objetos do *Oracle Data Integrator*: Comandos para interagir com objetos do próprio ODI.
- SAP: Alguns comandos para geração de arquivos no formato interno do sistema SAP.
- Utilitários: Possui comandos genéricos como executar um comando do sistema operacional ou efetuar uma operação de *kill* no agente do ODI.

A Figura 4.11 exibe o diagrama de execução do pacote PKG_NOTAS_FISCAIS. O componente “ReplicateNFHeader” é responsável pela execução do cenário de integração dos dados do “header” de notas fiscais. Que corresponde à execução da *interface* apresentada na Figura 4.9. No caso de falha de execução deste, será executado o componente “MailReplicateNFHeaderFail” onde será enviado um email de notificação de falha desta integração. No caso de sucesso o próximo passo é o comando “ReplicateNFLines” responsável pela integração dos dados de itens de notas fiscais. Ou seja, execução de uma *interface* responsável pela integração dos dados referentes a itens de notas fiscais. No caso de falha do mesmo será enviado um email de notificação de falha pelo componente “MailReplicateLinesFail”.

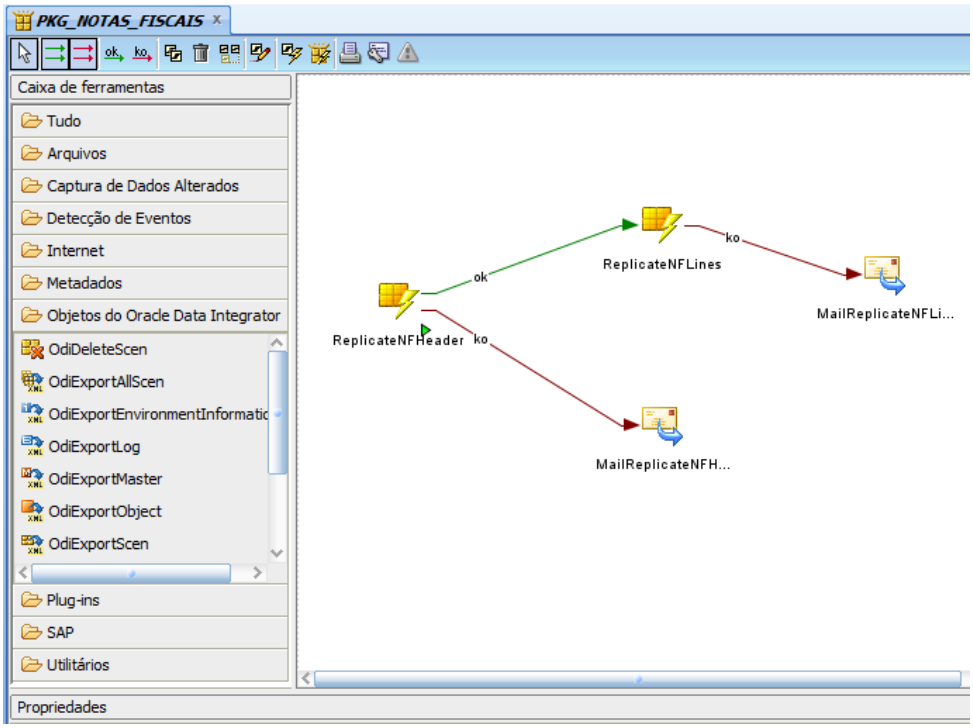

Figura 4.11 - Diagrama de execução do pacote

Para a realização dos testes deste trabalho não será necessário fazer uso dos comandos da *Open Tools*, no entanto estes são de grande utilidade no desenvolvimento de fluxos alternativos e controles de exceção de um módulo. A seguir são apresentados todos os experimentos realizados.

5 EXPERIMENTOS REALIZADOS

Antes de iniciar os experimentos no desenvolvimento da replicação com o *Oracle Data Integrator*, foi necessário preparar um ambiente para testes para avaliar a ferramenta. A criação de um ambiente de testes é necessária, pois não foi concedida permissão da empresa para realização de experimentos em ambiente de produção. Importante ressaltar que os resultados dos testes apresentados neste trabalho foram obtidos utilizando-se da versão 11g (11.1.1) do *Oracle Data Integrator*.

5.1 Ambiente de testes

Os testes foram realizados em dois cenários. Como existem diversos módulos de integração para o SGBD Oracle, inicialmente foram realizados testes locais na tentativa de identificar o módulo com o qual se atingirá melhor desempenho. Neste cenário tanto os servidores de dados como o servidor do ODI encontram-se no mesmo servidor físico. A Figura 5.1 ilustra este cenário, onde o ODI realizará a leitura dos dados no SGBD PostgreSQL, e a integração no SGBD Oracle. Com isso, será possível analisar as estatísticas de execução para cada módulo, a fim de optar pelo módulo mais performático.

Figura 5.1 - Cenário de teste local

O segundo cenário de testes utilizado por esse trabalho é composto por seis computadores, o servidor central, o servidor do ODI, e quatro máquinas clientes. A Figura 5.2 detalha a distribuição dos servidores do ambiente de testes. O servidor Oracle corresponde à base de dados central, enquanto que os servidores do PostgreSQL contêm a base de dados referente a uma filial. O servidor ODI deve ter acesso a todas as bases de dados, pois é responsável por verificar as alterações ocorridas em uma determinada base de dados e replicar esta para as demais.

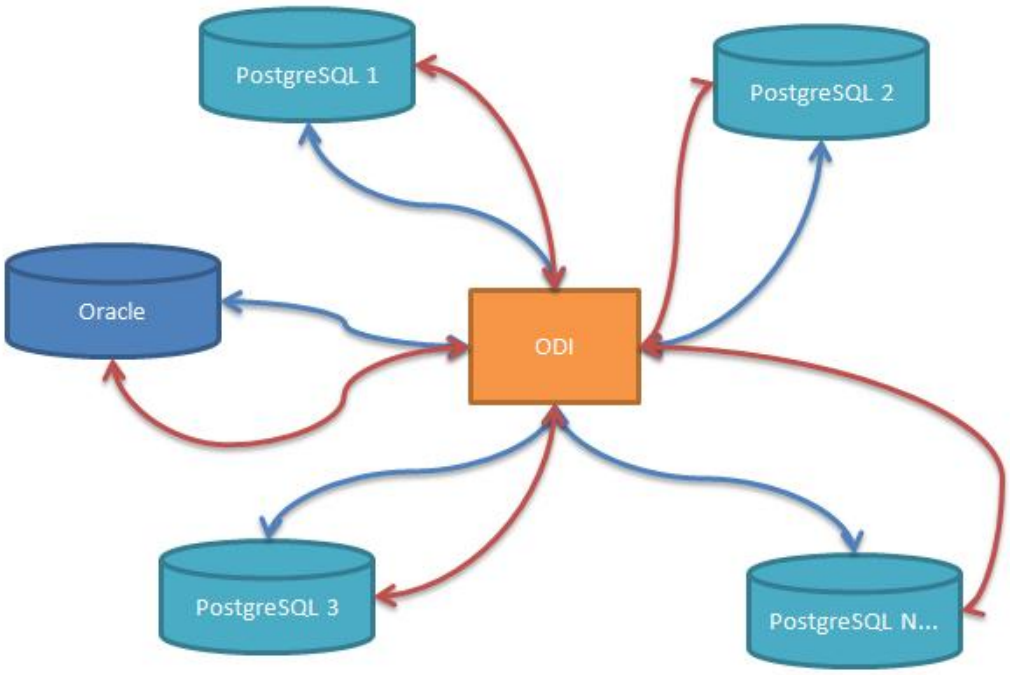

Figura 5.2 - Estrutura física do ambiente

A Tabela 5.1 apresenta as características de *hardware* e versão de SGBD instalado em cada servidor do ambiente.

Tabela 5.1 - Propriedades do ambiente de testes

Máquina	Memória	Processador	Disco Rígido	SGBD
Servidor Central	8 GB	I5 2.8GHz	500 GB	Oracle 11g
Servidor ODI	4 GB	I3 2.8GHz	500 GB	Oracle 10g
Cliente 1	8 GB	I5 2.8GHz	320 GB	PostgreSQL 9.0
Cliente 2	4 GB	2DUO 2.6GHz	500 GB	PostgreSQL 9.0
Cliente 3	3 GB	2DUO 2.4GHz	320 GB	PostgreSQL 9.0
Cliente 4	1 GB	Dual core 2.2GHz	320 GB	PostgreSQL 9.0

A máquina que contém o servidor do ODI também possui o SGDB Oracle. Nele serão criados os repositórios do ODI, tanto o repositório mestre, quanto o repositório de trabalho, onde serão desenvolvidos os módulos de integração.

Para o servidor principal foi utilizado o Oracle 11g, já para o servidor de repositórios do ODI pôde-se utilizar o Oracle 10g XE visto que não há necessidade de grande armazenamento de dados nos repositórios, para os módulos desenvolvidos nos experimentos deste trabalho. Todas as estações que representam as bases de dados de filiais foram configuradas com o SGBD PostgreSQL.

Com o ambiente de testes completamente instalado e configurado pôde-se então iniciar o desenvolvimento de projetos com o *Oracle Data Integrator*.

5.2 Módulos de integração desenvolvidos

Para a avaliação da ferramenta foram desenvolvidos os módulos de integração referentes às estruturas de produtos, clientes e faturamento, pois estas estão entre as maiores estruturas da base de dados. Além disso, estruturas de clientes e produtos são críticas. Em uma situação hipotética, uma informação desatualizada em uma das bases de dados para estas estruturas pode causar uma inconsistência no faturamento resultando em uma infração na legislação fiscal, tornando a empresa sujeita a uma multa por sonegação de imposto, por exemplo.

Tabela 5.2 - Estatísticas das estruturas replicadas

Estrutura	Quantidade de Registros	Quantidade de Colunas
Produtos	100572	82
Clientes	1207424	235
Notas Fiscais	306166	86
Itens de Notas Fiscais	747478	91

A Tabela 5.2 exibe algumas informações sobre as tabelas que foram utilizadas nos testes como a quantidade de registros e a quantidade de colunas existentes em cada tabela. As informações das estruturas de notas fiscais correspondem aos dados de apenas uma filial, enquanto que às de produtos e clientes correspondem ao volume total da empresa.

Inicialmente foram executados testes de replicação envolvendo apenas um servidor e um cliente, com a finalidade de avaliar primeiramente a replicação entre Oracle e PostgreSQL, em seguida avaliar os diferentes módulos de integração para Oracle a fim de identificar o módulo com melhor desempenho. Para estes testes foram utilizadas as tabelas de produtos e clientes, efetuando replicação completa da tabela, entre o SGBD de origem e o SGBD destino. Estes testes foram executados diversas vezes, e as estatísticas sobre execuções de módulos descritas foram obtidas pela média calculada com base em cinco execuções do referido módulo.

5.3 Avaliação da replicação entre Oracle e PostgreSQL

A Tabela 5.3 apresenta estatísticas da replicação da tabela de produtos entre duas bases de dados. Nestas estatísticas percebe-se que há um aumento elevado no tempo de

validação, que corresponde à execução do módulo CKM (responsável pelas validações de dados) para quando o SGBD destino é o PostgreSQL. Neste caso, a causa do elevado tempo para processar as validações ocorreu em função do comando SQL gerado por este CKM para validação de *primary keys* na tabela. Tal comando utiliza um filtro com uma restrição NOT EXISTS, mesmo método utilizado no SGBD Oracle, no entanto o desempenho deste no PostgreSQL foi bastante inferior, não sendo aconselhável a sua utilização para replicação de um grande volume de dados.

Tabela 5.3 - Replicação de produtos

Origem	Destino	Módulo de Integração	Validação PK	Tempo Carga	Tempo Validação	Tempo Integração	Tempo Total
PG	ORA	*Incremental Update*	Sim	10s	3s	27s	40s
ORA	PG	*Incremental Update*	Sim	23s	5089s	31s	5143s
PG	ORA	*Incremental Update*	Não	10s	3s	27s	40s
ORA	PG	*Incremental Update*	Não	22s	3s	24s	49s

5.4 Avaliação dos módulos de integração do Oracle

A Tabela 5.4 apresenta os resultados obtidos entre diferentes módulos de integração do ODI. Estes são os principais módulos de *Incremental Update* disponíveis para Oracle. Os KMs *Incremental Update* e *Incremental Update* (MERGE) tiveram sua execução concluída com tempos praticamente iguais, tendo uma variação de apenas 1 segundo. Já o módulo *Incremental Update* (PL/SQL) mostrou-se bastante lento em comparação com os demais. Tendo sua execução finalizada em um tempo no mínimo cinco vezes maior.

Tabela 5.4 - Comparação da execução entre módulos de integração de produtos

Origem	Destino	Módulo de Integração	Validação PK	Tempo Carga	Tempo Validação	Tempo Integração	Tempo Total
PG	ORA	*Incremental Update*	Sim	10s	3s	4s	17s
PG	ORA	*Incremental Update* (MERGE)	Sim	10s	3s	3s	16s
PG	ORA	*Incremental Update* (PL/SQL)	Sim	10s	3s	22s	35s

Módulos baseados em atualização *Control Append*, ou seja, que executam uma operação TRUNCATE nas tabelas do SGBD, não foram submetidos aos testes, pois não se enquadram no modelo de atualização que deve ser implantado. A execução de um TRUNCATE não é valida, pois as tabelas contêm referências ou *constraints*, que impedem a execução deste comando.

Como dois módulos apresentaram tempos bastante próximos na replicação de produtos, decidiu-se realizar um teste com maior volume de dados. Para isto foram realizados testes de replicação com a tabela de clientes, com a finalidade de avaliar o resultado de cada módulo.

A Tabela 5.5 apresenta os tempos de execução para a integração de dados de clientes para os diferentes módulos de conhecimento utilizados. Nesta situação pôde-se perceber uma variação nos tempos de execução dos KMs *Incremental Update* e *Incremental Update* (MERGE). O módulo *Incremental Update* obteve melhor tempo de execução, com uma diferença de 94 segundos para o módulo de atualização com MERGE. Já o módulo de atualização com PL/SQL comprovou ser o mais lento dos três módulos, finalizando a execução com 1604 segundos de atraso em comparação ao primeiro.

Tabela 5.5 - Tempos de replicação de clientes

Origem	Destino	Módulo de Integração	Validação PK	Tempo Carga	Tempo Validação	Tempo Integração	Tempo Total
PG	ORA	*Incremental Update*	Sim	292s	14s	151s	457s
PG	ORA	*Incremental Update* (MERGE)	Sim	292s	14s	245s	551s
PG	ORA	*Incremental Update* (PL/SQL)	Sim	292s	14s	1449s	1755s

Pelos resultados obtidos nos testes locais, o módulo *Incremental Update*, foi o que obteve menor tempo de execução. Portanto os testes de replicação envolvendo mais de uma base de dados serão realizados utilizando apenas este módulo. No entanto, o tempo de execução de cada atualização depende tanto do *hardware* de cada servidor como da capacidade de tráfego de dados na rede.

A Tabela 5.6 apresenta os tempos de execução em um cenário com vários servidores. Neste percebe-se que o SGBD Oracle manteve baixos tempos de execução, enquanto que o SGBD PostgreSQL apresenta tempos mais elevados. Além das replicações serem executadas em maior tempo no PostgreSQL, a situação se agrava quando é ativada a opção de validação de erros de *primary keys*.

Tabela 5.6 - Replicação de vários servidores utilizando o módulo *incremental update*

Origem	Destino	Filial	Validação PK	Tempo Carga	Tempo Validação	Tempo Integração	Tempo Total
ORA	PG	1	Não	22s	2s	87s	111s
ORA	PG	2	Não	215s	5s	79s	299s
ORA	PG	3	Não	201s	4s	186s	391s

ORA	PG	4	Não	231s	10s	512s	753s
PG	ORA	1	Sim	18s	1s	2s	24s
PG	ORA	2	Sim	48s	1s	2s	52s
PG	ORA	3	Sim	56s	2s	2s	59s
PG	ORA	4	Sim	68s	4s	2s	71s
ORA	PG	1	Sim	22s	4991s	36s	5049s
ORA	PG	2	Sim	290s	6691s	42s	7023s
ORA	PG	3	Sim	304s	11716s	50s	12070s
ORA	PG	4	Sim	300s	12942s	242s	13484s

O PostgreSQL comprovou processar as consultas de validação de dados com grande dificuldade, em comparação ao Oracle que sempre as processou em poucos segundos. No caso de uma replicação pelo método de *journalizing*, não há mudanças nos tempos de execução do módulo, pois a única diferença no módulo é que o *Oracle Data Integrator* irá verificar os dados a serem replicados em sua estrutura interna, criada com a finalidade de registrar as alterações na origem dos dados.

A estrutura de notas fiscais foi submetida aos testes com *journalizing*. Quando um módulo de replicação é desenvolvido com módulos de JKM, o ODI automaticamente cria no banco de dados, a estrutura para detecção dos dados alterados. Essa estrutura é composta por tabelas, visões e *triggers*.

Neste cenário de *journalizing*, os dados são constantemente replicados à medida que sofrem alterações na origem. Esse tipo de recurso não afetou o desempenho. Testes realizados aplicando alterações em toda a tabela mantiveram a mesma média de execução de um módulo normal.

5.5 Proposta de solução

O modelo de replicação proposto por este trabalho baseia-se no modelo utilizado nos testes. Neste utiliza-se uma máquina dedicada para o *Oracle Data Integrator*. É importante ser uma máquina dedicada para que outros serviços não comprometam os recursos do sistema, bem como a operação do próprio ODI, visto que uma falha no sistema de replicação é um problema crítico.

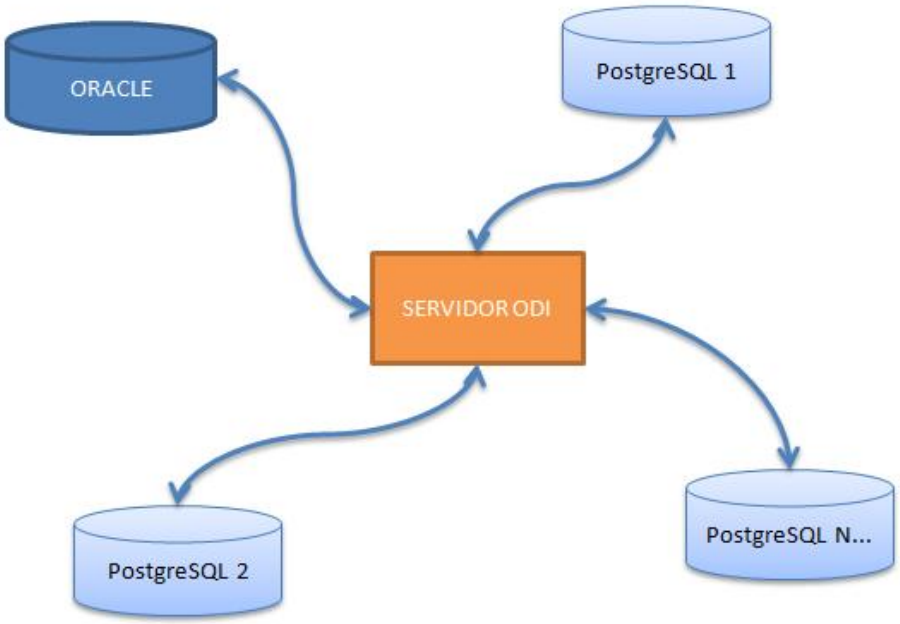

Figura 5.3 - Modelo proposto

A Figura 5.3 apresenta o modelo do ambiente. O servidor ODI é dedicado, e este é responsável por ler as informações a serem replicadas de uma determinada base de dados, realizando a atualização na base de dados destino.

Quando aos módulos de integração para Oracle a serem utilizados, sugere-se a opção dos módulos *Incremental Update*, pois de acordo com as avaliações dos testes já apresentados, estes foram os módulos que comprovaram melhor desempenho.

5.6 Aspectos positivos

A replicação entre bases de dados heterogêneas foi realizada com grande naturalidade com o ODI. Como já anunciado pela Oracle, o *Oracle Data Integrator* pôde realizar a integração de dados entre SGBD distintos com facilidade. Além disso, não está restrita apenas aos utilizados neste trabalho, como também está preparada para possíveis novos SGBD no ambiente, podendo também ser utilizada para desenvolver módulos de extração ou importação de dados em arquivos.

O gerenciamento dos módulos de integração de forma independente oferece facilidades em momentos críticos ou na ocorrência de uma falha. Na ocorrência de um incidente em um módulo de integração, não é necessário interromper todos os módulos em execução. No contexto atual da empresa, o gerenciador de replicação é global, na ocorrência de um incidente todas as integrações são interrompidas.

Outra vantagem é a execução paralela dos módulos sendo possível definir prioridades e frequências de atualização distintas a diferentes estruturas da base de dados. Isso é importante, pois no contexto da organização existem estruturas que deveriam ser atualizadas quase que simultaneamente, e outras que poderiam ser atualizadas uma vez ao dia. No cenário atual da empresa não há como priorizar atualizações específicas. Uma atualização aguarda em uma fila, até que todas as suas precedentes tenham sido concluídas.

Além das vantagens já mencionadas, o ODI oferece uma interface de desenvolvimento bastante simples, proporcionado grande facilidade e velocidade à criação e manutenção de módulos. Podendo ainda ser utilizado no desenvolvimento de rotinas para geração de arquivos, substituindo rotinas complexas de formatação e geração de arquivos do sistema de origem.

5.7 Aspectos negativos

Sendo uma ferramenta da Oracle, é compreensível que esta possua grande compatibilidade de recursos para com este banco de dados. No entanto, para produtos de outros fornecedores a utilização da ferramenta pode não ser tão positiva quanto para o Oracle.

No caso do PostgreSQL que é a outra base de dados do ambiente, não existe um módulo JKM para realizar o *journalizing* nesta base de dados. Sendo assim, torna-se necessário investir no desenvolvimento de um novo módulo de conhecimento para realizar esta atividade, ou adaptar um módulo existente de outro SGBD.

Outro problema encontrado no PostgreSQL é que este não processa as consultas com a mesma eficiência que o SGBD Oracle. Foram apresentadas nos testes algumas situações de validação de *primary keys*, onde o Oracle conseguia processar em poucos segundos enquanto o PostgreSQL necessitava de vários minutos. Nesta situação, caso a validação de *primary keys* se essa validação fosse mesmo importante, seria necessário customizar o módulo de conhecimento utilizado na validação no banco de dados PostgreSQL.

5.7.1 Alternativa à falta de módulo JKM no PostgreSQL

Uma alternativa a falta de módulos de *journalizing* para o PostgreSQL seria a utilização de um campo de controle na tabela de origem. Este campo controlaria o estado da replicação. Nesta solução existem basicamente três estados “NÃO PROCESSADO”, “LIDO”, “PROCESSADO”.

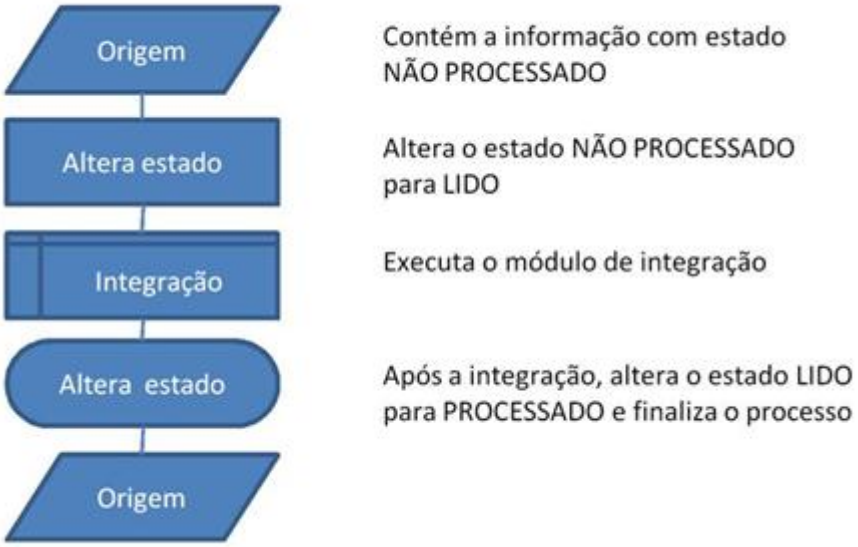

Figura 5.4 - Fluxo de integração com controle de estado

A Figura 5.4 apresenta o fluxo de execução em um processo de integração com controle de estado. Neste processo os dados manipulados através do sistema de origem são gravados em um estado "NÃO PROCESSADO". O ODI antes de iniciar o processo de integração altera o estado de todos os registros "NÃO PROCESSADO" para "LIDO". Após a mudança de estado, a integração será executada considerando apenas os registros em estado "LIDO". Com o término da integração todos os registros em estado "LIDO" são atualizados para "PROCESSADO".

Com o processo baseado em um campo de estado, basta agendar a execução do módulo ODI em intervalos de tempo. Com isso a cada intervalo todos os registros que foram modificados na origem serão replicados para o banco de dados destino. Esta solução exige uma adequação no SGBD de origem em função do campo de controle de estado, portanto pode não ser a mais indicada.

CONCLUSÃO

Implantar um sistema de replicação de dados pode não ser uma tarefa complexa. No entanto, garantir o desempenho e a escalabilidade deste processo torna-se complexo, à medida que os negócios da empresa expandem em um planejamento de infra-estrutura.

Conforme visto, a empresa conta com uma ferramenta para replicação de dados que não tem atendido as necessidades da empresa. Atualmente, o processo de replicação está comprometido pelo desempenho do gerenciador de replicação, e em algumas ocasiões há até mesmo ocorrências de réplicas inconsistentes em função de falhas ocorridas no momento das atualizações.

Pôde-se perceber também que o Oracle dispõe de diversos recursos para aplicação de replicação. No entanto como há necessidade de replicar dados entre bases de dados distintas, no caso Oracle e PostgreSQL, faz-se necessária a utilização de uma ferramenta específica para tal. O *Oracle Data Integrator* mostrou-se como a ferramenta mais adequada para o cenário da empresa, tanto pelos recursos anunciados pela ferramenta, quanto pela quantidade de documentação, embora a utilização de *streams* também possa ser uma solução interessante para o problema.

É importante ressaltar que a qualidade da replicação não depende exclusivamente da qualidade da ferramenta utilizada para este processo, e da quantidade de recursos que esta possui. Depende essencialmente da maneira como essa rotina é implementada, se o modo de replicação executado é o que melhor atende as necessidades, bem como, se a estrutura física e de rede estão aptas para atender este processo.

Com base nos problemas de replicação de dados encontrados no cenário da empresa, o objetivo deste trabalho foi apresentar uma proposta de solução para o mesmo. Buscando não apenas a solução dos problemas enfrentados no cotidiano da empresa, bem como novos recursos que proporcionem um gerenciamento mais dinâmico, oferecendo maiores facilidades, tanto ao desenvolvimento e manutenção de rotinas de replicação, quanto à gerência e monitoramento do processo.

A maior dificuldade encontrada no desenvolvimento deste trabalho foi o fato de não ser possível realizar os testes de replicação no ambiente de produção, ou em um ambiente mais próximo da realidade. Para que os resultados destes apresentassem informações mais precisas dos prós e contras da utilização desta ferramenta.

Apesar das vantagens apresentadas por este trabalho, alguns aspectos dos ODI ainda podem ser aperfeiçoados. A falta de um módulo de *journalizing* para PostgreSQL é uma carência que pode ser suprida explorando-se o desenvolvimento de módulos de integração. Além disso, como exposto por este trabalho os módulos de validação apresentam algumas deficiências em certos pontos de execução. Assim, a exploração dos módulos de conhecimento pode agregar mais valor a esta ferramenta.

REFERÊNCIAS BIBLIOGRÁFICAS

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas distribuídos conceitos e projeto**. Porto Alegre: Bookman, 2007.

DATE, C.J.. **An introduction to database systems**. Pearson Education, Inc. 2004.

ELMASRI, Ramez; NAVATHE, Shamkant B.. **Fundaments of database systems**. Pearson Education, Inc. 2004.

ORACLE CORPORATION. **Oracle database advanced replication.** 2003.

ORACLE CORPORATION. **Oracle database 11g release 2 documentation.** 2010. Disponível em: <http://www.oracle.com/pls/db112/homepage> Acesso em: 05 out. 2010.

ORACLE CORPORATION. **Oracle Data Integrator documentation.** 2010. Disponível em: <http://download.oracle.com/docs/cd/E14571_01/odi.htm> Acesso em: 01 nov. 2010.

ORACLE CORPORATION. **Oracle GoldenGate documentation.** 2010. Disponível em: <http://download.oracle.com/docs/cd/E18101_01/index.htm> Acesso em: 05 nov. 2010.

ORACLE CORPORATION. **Oracle Streams An Oracle White Paper.** 2002. Disponível em: < http://www.oracle.com/technetwork/testcontent/streams-oracle-streams-twp-128298.pdf> Acesso em 10 dez. 2010.

ORACLE CORPORATION. **The Oracle data integrator enterprise edition architecture.** Fev. 2009. Disponível em: < http://www.oracle.com/technetwork/middleware/data-integrator/overview/oracledi-architecture-1-129425.pdf > Acesso em: 20 out. 2010.

ORACLE CORPORATION. **Oracle® Fusion Middleware Developer's Guide for Oracle Data Integrator 11g Release 1 (11.1.1).** 2010. Disponível em: <http://download.oracle.com/docs/cd/E14571_01/integrate.1111/e12643/data_capture.htm> Acesso em: 15 abr. 2011.

ORACLE CORPORATION. **Oracle Data Integrator Tools Reference Manual**. 3 Release. Oracle, USA, 2009.

ÖZSU, M. Tamer; VALDURIEZ, Patrick. **Princípios de sistemas de bancos de dados distribuídos**. Rio de Janeiro: Campus, 2001.

POSTGRESQL. **PostgreSQL 9.0.4 Documentation**. 2010. Disponível em: <http://www.postgresql.org/docs/9.0/interactive/index.html> 2010. Acesso em: 10 out. 2010.

POSTGRESQL. **PostgreSQL Streaming Replication**. 2010. Disponível em: <http://wiki.postgresql.org/wiki/Streaming_Replication> Acesso em: 22 jun. 2011.

RAMAKRISHNAN, Raghu; GEHRKE, Johannes. S**istemas de gerenciamento de banco de dados.** São Paulo: McGraw-Hill, 2008.

SILBERCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S.. **Sistema de banco de dados.** 3. Ed. São Paulo: Pearson Makron Books, 1999.

WIECK, Jan**; SLONY-I A replication system for PostgreSQL**. 2010. Disponível em: <http://developer.postgresql.org/~wieck/slony1/Slony-I-concept.pdf> Acesso em: 22 jun. 2011.

SLONY documentation. 2009. Disponível em: <http://slony.info/documentation/> Acesso em: 10 out. 2010.

SLONY documentation 2.0. 2010. Disponível em: < http://slony.info/documentation/2.0/> Acesso em: 10 out. 2010.

TUMMA, Madhu. **Oracle streams high speed replication and data sharing.** North Carolina: Rampant TechPress, 2005.