FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CARLOS JESUS DE OLIVEIRA RODRIGUES

ARMAZENAMENTO E RECUPERAÇÃO DE IMAGENS MÉDICAS UTILIZANDO O SGBD ORACLE9i DISTRIBUÍDO

MARÍLIA
2005

CARLOS JESUS DE OLIVEIRA RODRIGUES

ARMAZENAMENTO E RECUPERAÇÃO DE IMAGENS MÉDICAS UTILIZANDO O SGBD ORACLE9i DISTRIBUÍDO

Monografia apresentada ao Curso de Ciência da Computação, da Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Profª. Dra. Fátima L. S. Nunes

MARÍLIA
2005

CARLOS JESUS DE OLIVEIRA RODRIGUES

ARMAZENAMENTO E RECUPERAÇÃO DE IMAGENS MÉDICAS UTILIZANDO O SGBD ORACLE9i DISTRIBUÍDO

BANCA EXAMINADORA DA MONOGRAFIA PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

CONCEITO FINAL: _____(_____)

ORIENTADOR: _____

Profª Dra. Fátima L. S. Nunes

1º EXAMINADOR: _____

Prof. Dr.

2º EXAMINADOR: _____

Prof. Dr.

Marília, _____de _____de 2005

RODRIGUES, Carlos Jesus de Oliveira. Armazenamento e Recuperação de Imagens Médicas Utilizando o SGBD Oracle9i Distribuído.
Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

RESUMO

Este trabalho tem por objetivo apresentar o SGBD Oracle9i como sistema de banco de dados distribuído para o armazenamento e recuperação de imagens médicas, realizando uma comparação entre um tipo de dado que armazena a imagem em sua forma binária e outro que armazena um conjunto de caracteres que representa o local (*path*) no qual a imagem está armazenada. Para atingir o objetivo proposto, foram necessárias a instalação e configuração do Oracle9i Distribuído e a criação de ferramentas para execução de testes de performance de armazenamento e recuperação da imagem pelos diversos *hosts* do SBDD.

Palavras-chave: Sistemas de Banco de Dados Distribuídos(SBDD), SGBD Oracle9i, Sistema Gerenciador de Banco de Dados(SGBD), Armazenamento e Recuperação de Imagens.

RODRIGUES, Carlos Jesus de Oliveira. Armazenamento e Recuperação de Imagens Médicas Utilizando o SGBD Oracle9i Distribuído.
Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

ABSTRACT

This work has for objective to present the SGBD Oracle9i as distributed database system for the storage and recovery of medical images, carrying through a comparison between a type of data that store the image in its binary form, and another one that stores a set of characters that it represent the place in which the image is stored. To reach the considered objective, it was necessary the implementation of the Distributed Oracle9i and creation of tools for execution of tests of storage performance and recovery of the image for diverse node of the SBDD.

Keywords: Distributed Database System (DDS), DBMS Oracle9i, Distributed Database Management Systems (DDBMS), Storage and Recovery of Images
.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

INTRODUÇÃO

Com o avanço da tecnologia e dos sistemas distribuídos, grandes corporações estão migrando de uma arquitetura centralizada para uma arquitetura distribuída. Para tanto, sistemas de Banco de Dados Distribuídos estão sendo utilizados para a substituição dos sistemas de Banco de Dados centralizados por oferecem uma melhor disponibilidade das informações armazenadas e confiabilidade operacional.

Centros médicos necessitam de agilidade e segurança no acesso ao prontuário do paciente, portanto sistemas para o armazenamento e recuperação de imagens médicas (PACS) estão sendo implantados em hospitais e clínicas sob o paradigma de sistemas de Banco de Dados Distribuído, oferecendo um melhor gerenciamento do grande volume de dados gerados pelos centros médicos (NETO et al, 2005).

O Sistema Gerenciador de Banco de Dados (SGBD) deve apresentar características imprescindíveis ao desenvolvimento de tais sistemas em ambientes médicos.

Objetivo

Este trabalho tem por objetivo avaliar o SGBD Oracle9i distribuído para armazenamento e recuperação de imagens médicas, utilizando dois tipos de dados diferentes disponibilizados pelo SGBD Oracle9i. O primeiro armazena a imagem no Banco de Dados em sua forma binária e o outro consiste em um atributo do tipo *string* que registra apenas o local no qual a imagem está fisicamente armazenada.

Organização da monografia

Esta monografia está organizada da seguinte forma:

O Capítulo 1 descreve-se o conceito de Sistemas de Banco de Dados de Distribuídos, mostrando as técnicas de fragmentação, replicação e alocação dos dados pelos diversos *hosts* que constituem a rede.

No Capítulo 2 são apresentados os conceitos sobre sistema que estão sendo implantados em hospitais e centros médicos a fim de proporcionar o gerenciamento das imagens médicas que geram um grande volume de dados. Apresenta ainda conceitos sobre os diversos formatos de imagens digitais.

No Capítulo 3 apresenta-se como procede a comunicação do SGBD Oracle9i em um Sistema de Banco de Dados Distribuído, descrevendo cada componente necessário para a distribuição dos dados.

No Capítulo 4 são demonstrados os procedimentos utilizados para configuração do SGBD Oracle9i para que o mesmo opere de forma distribuída.

No Capítulo 5 descrevem-se as ferramentas desenvolvidas para operar no sistema de Banco de Dados Distribuído com a finalidade de fornecer os dados para análise e comparação.

O Capítulo 6 apresentam-se os valores obtidos, comparados e representados graficamente para a obtenção da conclusão desta monografia.

A conclusão da monografia é apresentada no Capítulo 7 e também são citadas as dificuldades encontradas durante toda a fase do projeto.

1 - BANCOS DE DADOS DISTRIBUÍDOS

Um sistema de computação distribuído consiste em um conjunto de computadores autônomos e não necessariamente homogêneos, que utilizam um sistema de rede de comunicação para realizar determinadas tarefas a eles atribuídas. Esses sistemas têm como objetivo fragmentar os grandes e complexos problemas, não gerenciáveis, em fragmentos menores, e resolver o mesmo de maneira eficiente e coordenada.

Özsu e Valduriez(2001) definem um Sistema de Banco de Dados Distribuídos (SBDD) como sendo uma coleção de vários Bancos de Dados logicamente inter-relacionados, distribuídos por uma rede de computadores, e define um Sistema Gerenciador de Banco de Dados Distribuídos (SGBDD) como sendo o *software* que disponibiliza o gerenciamento do banco de dados distribuído e que proporciona a distribuição transparente para o usuário.

Um SBDD é, portanto, um conjunto de Banco de Dados, na qual cada *host* da rede possui o seu próprio Banco de Dados, e estes são interconectados logicamente aparentando ser uma única base de dados. Um *host* pode acessar e modificar os dados simultaneamente nos vários Bancos de Dados existentes no sistema.

1.1 Promessas dos Sistemas de Bancos de Dados Distribuídos

Em sistemas centralizados geralmente o *host* central é um computador de grande porte (*mainframe*), podendo fazer com que a relação custo/desempenho torne-se inviável, pois computadores de grande porte possuem um custo elevado em comparação com computadores de médio e pequeno porte, que podem proporcionar um melhor desempenho com menor custo. Portanto, pode-se obter um melhor desempenho com um menor gasto; essa economia é uma das razões para a descentralização.

Outras questões como a distribuição e a replicação dos dados de forma transparente para o usuário, confiabilidade e disponibilidade na distribuição dos dados, otimização do sistema e capacidade de expansibilidade sem degradação do desempenho do sistema, são razões para a utilização de SBDD (ÖZSU e VALDURIEZ, 2001)(ELMASRI e NAVATHE, 2005).

1.1.1 Gerenciamento de dados distribuídos e replicados com diferentes tipos de transparências

Em termos ideais, um SGBD é transparente na distribuição se este fornece a ocultação das informações de mais baixo nível de implementação, ou seja, esconde a localização física do armazenamento de cada objeto (tabela ou relação), fazendo o sistema parecer um simples sistema de Banco de Dados (ÖZSU e VALDURIEZ, 2001) (ELMASRI e NAVATHE, 2005).

As transparências possíveis são:

- **Transparência de rede ou de distribuição**: consiste em tornar a rede oculta para os usuários. Pode ser dividida entre Transparência de Localização e Transparência de Nomenclatura:
    - **Transparência de localização**: refere-se à ocultação da localização física dos objetos do banco de dados.
    - **Transparência de nomenclatura**: especifica que cada objeto do Banco de Dados tem um domínio exclusivo.
- **Transparência de replicação:** refere-se à existência de dados replicados que podem estar armazenados em várias máquinas da rede para melhor

disponibilidade, desempenho e confiabilidade. A transparência de replicação faz com que o usuário não tenha o conhecimento da existência de réplicas.

- **Transparência de fragmentação: e**stá diretamente ligada ao processamento de consultas e ao desempenho do sistema. Uma consulta global feita pelo usuário deve ser transformada em diversos fragmentos de consultas, pois estas consultas devem ser baseadas nos fragmentos e não nas relações. A transparência de fragmentação proporciona ao usuário a abstração de fragmentos fazendo com que o sistema pareça centralizado.

O principal objetivo da transparência é fornecer a independência de dados ao sistema, seja ela física (descrição física dos dados pode mudar sem afetar as aplicações do usuário) ou lógica (mudanças na estrutura lógica do banco de dados não afetarão as aplicações do usuário).

1.1.2 Confiabilidade e disponibilidade na distribuição de dados

Devido à distribuição dos dados e dos SGBDs por diversos *hosts* da rede, pode ocorrer falhas em alguns locais. A falha de um ou mais *hosts* não torna o sistema inoperante. Os dados que estão armazenados nos *hosts* inoperantes podem ser localizados em outros *hosts* participantes do sistema. Isso garante a eliminação de pontos únicos de falha nesses sistemas. Portanto, a distribuição permite que os dados se localizem mais perto de onde são solicitados ou onde são gerados. Isso melhora tanto a confiabilidade quanto a disponibilidade (ÖZSU e VALDURIEZ, 2001).

O SGBDD deve garantir a integridade do banco de dados e a consistência dos dados que estão localizados nos *hosts* que operam no sistema, como também no *host* de falha após sua recuperação. A atomicidade é questão de extrema importância para o projeto de sistema de Banco de Dados, pois as transações podem levar o Banco de Dados distribuído a um estado

inconsistente. Porém, a atomicidade implica em que uma transação deve ser executada por um todo, ou esta não deverá ser executada, ou seja, deve garantir a propriedade do tudo ou nada na execução de transações (ÖZSU e VALDURIEZ, 2001).

Outros aperfeiçoamentos são conseguidos por meio de criteriosa replicação de dados e software pelos diversos *hosts*.

1.1.3 Desempenho otimizado

Um SGBDD fragmenta o Banco de Dados mantendo os dados mais próximos de onde são mais requisitados (por questão de sobrecarga de processamento). A localização de dados reduz a disputa entre serviços da CPU e de E/S (Entrada/Saída) e, ao mesmo tempo, reduz a demora no acesso remoto que sistemas de redes de longa distância (*wide-area network - WAN)* proporcionam (ÖZSU e VALDURIEZ, 2001).

O paralelismo inerente de sistemas distribuídos deve ser explorado, proporcionando várias consultas ao mesmo tempo (paralelismo entre consultas) e a divisão de consultas em sub-consultas, com cada uma sendo executada em paralelo, em *hosts* diferentes do banco de dados distribuído (paralelismo intraconsultas). Explorando essa propriedade há um ganho no processamento devido ao menor número de consultas executadas em cada *host* (ÖZSU e VALDURIEZ, 2001).

1.1.4 Facilidade na expansão do sistema

A inclusão de novos elementos ao sistema proporciona a melhora no poder computacional (crescimento incremental), aumento no tamanho dos bancos de dados, e aumento da capacidade da rede. No aspecto econômico, a expansão é conseguida de maneira fácil nesse tipo de sistema.

1.2 Funções adicionais dos bancos de dados distribuídos

Para conseguir seu pleno potencial, o software do SGBDD deve fornecer os aspectos anteriormente citados somados aos aspectos de SGBD centralizados. Isso torna o ambiente distribuído mais complexo que o sistema centralizado. Essa complexidade nos sistemas distribuídos ocasiona novos problemas que são influenciados por três fatores (ÖZSU e VALDURIEZ, 2001).

O primeiro fator é a replicação dos dados no ambiente distribuído. A replicação pode proporcionar que um banco de dados distribuído seja projetado de forma que ele resida por inteiro, ou parte dele, em diferentes *hosts* de uma rede de computadores. Portanto, não é necessário que todo *host* contenha o banco de dados, mas a existência de mais de um *host* com o banco de dados é de extrema importância. Esta possível duplicação de dados está principalmente ligada às considerações de confiabilidade e de eficiência citadas anteriormente. Logo, o SGBD é responsável por escolher, em caso de leitura, qual cópia acessar e, em caso de atualizações, estas devem refletir-se em todas cópias.

O segundo fator é que, por ventura de falhas em algum *host* local (devido a problemas de hardware ou software) ou por falhas nos meios de comunicação durante a execução de uma atualização, o sistema deve garantir que o efeito se refletirá nos dados do *host* que esteve em queda ou nos *hosts* inacessíveis, imediatamente após sua recuperação.

O terceiro ponto é que devido cada *host* sofrer uma carência de informações instantâneas sobre as ações que ocorrem nos outros *hosts* no mesmo instante, a sincronização de transações em vários *hosts* se torna mais complexa em relação a sistemas centralizados.

O SGBDD, para conseguir seu pleno potencial, deve fornecer os aspectos citados na Seção 2.1, como também deve ser capaz de oferecer as seguintes funções:

- **Processamento de consultas distribuídas:** baseia-se na utilização de estratégias para otimização do local onde é utilizado o paralelismo inerente,

devido a custos de processamento local, custo de comunicação, fatores de distribuição de dados e a falta de informação suficiente disponível no local (ÖZSU e VALDURIEZ, 2001) (DATE, 2004) (ELMASRI e NAVATHE, 2005).

- **Gerenciamento de diretório (catálogo) distribuído**: Informações referentes à estrutura e à localização dos dados armazenados estão contidos no catálogo ou diretório. Este catálogo pode estar centralizado em um único local ou distribuído pelos vários *hosts* que terão uma cópia que deverá estar sempre atualizada (ÖZSU e VALDURIEZ, 2001) (DATE, 2004) (ELMASRI e NAVATHE, 2005).
- **Gerenciamento de transações distribuídas:** Devem proporcionar estratégias de execução para consultas e transações que acessam dados de vários *hosts*, e para sincronizar o acesso aos dados distribuídos e para manter a integridade do Banco de Dados global (ÖZSU e VALDURIEZ, 2001) (DATE, 2004) (ELMASRI e NAVATHE, 2005).
- **Gerenciamento de dados replicados:** Devido à existência de várias réplicas de dados, o SGBDD é responsável por escolher uma das cópias armazenadas em caso de recuperação dos dados e assegurar que o efeito de uma atualização se refletirá em toda e qualquer cópia do banco de dados (ÖZSU e VALDURIEZ, 2001) (DATE, 2004) (ELMASRI e NAVATHE, 2005).
- **Segurança:** Deve assegurar o gerenciamento da segurança adequado e dos privilégios de autorização/acesso dos usuários, na execução de transações distribuídas (ÖZSU e VALDURIEZ, 2001) (DATE, 2004) (ELMASRI e NAVATHE, 2005).

- **Recuperação de banco de dados distribuídos:** deve fornecer habilidade para se recuperar de falhas ocorridas em algum *host* e na comunicação entre estes *hosts* (ÖZSU e VALDURIEZ, 2001) (DATE, 2004) (ELMASRI e NAVATHE, 2005).

1.3 Arquitetura de SGBDs distribuídos

A arquitetura trata da estruturação do sistema, ou seja, da identificação dos diversos componentes do sistema, definição das funcionalidades de cada componente e dos relacionamentos e interações entre eles. O principal objetivo da arquitetura é o fornecimento das funcionalidades vistas na Seção 2.1, enfatizando que na arquitetura do sistema distribuído deverá ser especificado o nível de transparência desejada para o sistema.

1.3.1 Arquitetura de Referência

Özsu e Valduriez (2001) apresentam o modelo arquitetônico ANSI/SPARC (*American National Standards Institute/Systems Planning and Requirements Committee*) como um modelo de referência utilizado para a padronização de sistemas de banco de dados, que tem como base a organização dos dados, dividindo em três níveis de esquema: interno, conceitual e externo, sendo que esta terminologia visa atingir as características (isolamento, integração e compartilhamento) de sistemas de banco de dados:

- *O nível interno* relaciona-se à forma como são realmente armazenados os dados (definição física e organização dos dados).
- *O nível externo* relaciona-se à forma como os usuários individuais visualizam o banco de dados (nível mais próximo aos usuários).

- *O nível conceitual* é o "nível de simulação" entre os dois anteriores. Pode ser considerado como a visão do grupo de usuários ou como a representação abstrata do Banco de Dados em sua totalidade (implementação através de tabelas e seus relacionamentos).

Os mapeamentos indicam de que forma estão relacionados os dados, ou seja, executa as transformações necessárias de modo que uma relação definida em um nível possa ser obtida em outro. Portanto, o mapeamento é quem possibilita a interação entre os níveis de visões. O dicionário/diretório de dados é quem possibilita os mapeamentos entre os níveis de visões. Este dicionário é um Banco de Dados que mantém informações (metadados) armazenadas sobre os dados reais armazenados no Banco de Dados (ÖZSU e VALDURIEZ, 2001).

1.3.2 Fatores para classificações de SGBDS distribuídos

No contexto de Sistemas de Banco de Dados Distribuídos existe uma distribuição de dados e software por diversos *hosts* conectados por uma rede de comunicação. Özu e Valduriez (2001) descrevem três fatores para a classificação de SGBDD, autonomia, distribuição e heterogeneidade.

1.3.2.1 Autonomia

A autonomia implica no grau de controle da administração dos dados de cada *host* que participa do sistema distribuído. Isso implica que cada *host* deve ser independente de forma a executar e gerenciar suas operações locais sobre seus dados sem que seja necessário consultar outros *hosts*. As operações originadas no mesmo *host* são ditas locais e as demais são chamadas de globais. Cada *host* local não depende de um outro *host* para obter seu pleno

funcionamento, ou seja, a descentralização, tanto dos dados quanto de controle, é uma necessidade nesse sistema.

Cada *host* é capaz de reter um grau de controle sobre os dados armazenados localmente. Em um sistema distribuído, existe um administrador global do Banco de Dados responsável pelo sistema inteiro. Uma parte dessas responsabilidades é delegada ao administrador do Banco de Dados local em cada *host* (SILBERCHATZ et al, 1999).

A taxonomia proposta por Özsu e Valduriez (2001) apresenta uma visão, considerando a existência de três tipos de autonomia, integração estreita (ou fortemente integrados), semi-autonomia e isolamento total (ou total autonomia). Estas correspondem, respectivamente, a três tipos de sistemas:

- Sistemas fortemente integrados: uma visão única de toda a base de dados é disponibilizada aos usuários compartilhadores da informação que pode estar fisicamente armazenada em múltiplas bases de dados.
- Sistemas semi-autônomos: consistem de *hosts* que podem operar independentemente, mas que participam de uma federação para permitir o compartilhamento de seus dados.
- Sistemas totalmente isolados: os componentes (*hosts*) individuais são SGBDs [1]*stand-alone* que desconhecem a existência de outros SGBDs, bem como a maneira de se comunicar com eles.

1.3.2.2 Distribuição

Este fator diz respeito à distribuição física dos dados por vários *hosts* da rede, permitindo que estes se localizem onde são gerados ou mais utilizados, e permitindo o compartilhamento desses dados com os demais *hosts*.

[1] Estações com autonomia de processamento que operam sozinhas.

Segundo Özsu e Valduriez (2001) existem três maneiras para se distribuir os dados: de forma centralizada, descentralizada (distribuído) e nenhuma distribuição. Na forma centralizada surge a arquitetura Cliente/Servidor; na descentralizada, a arquitetura não hierárquica. E, por fim, os sistemas de vários Bancos de Dados não apresentam distribuição.

A distribuição é um fator que implica na disponibilidade e confiabilidade citadas anteriormente na Seção 2.1.2.

1.3.2.3 Heterogeneidade

A heterogeneidade é um fator de classificação em relação às diferenças nos aspectos como *harware*, *software*, protocolos de comunicação, e podem utilizar diferentes sistemas de gerenciamento de dados. Os SGBDs podem se diferenciar nos aspectos de modelo de dados, linguagem de consulta e algoritmos de gerenciamento de transações. Esses SBDD são denominados sistemas de vários Bancos de Dados (*multidatabase*) ou sistemas de Banco de Dados Distribuídos Heterogêneos (ÖZSU e VALDURIEZ, 2001).

1.3.3 Arquitetura de SGBDs Distribuídos

Nesta seção são mostradas as principais arquiteturas existentes para sistemas distribuídos: Cliente/Servidor, SGBDs Distribuídos não hierárquicos e sistemas de vários Bancos de Dados (ÖZSU e VALDURIEZ, 2001).

1.3.3.1 Cliente/Servidor

A computação Cliente/Servidor pode ser considerada como uma arquitetura de dois níveis, em que estes níveis devem ser classificados como Clientes e Servidores, levando em consideração aspectos de funcionalidade a serem fornecidos.

O servidor oferece ao sistema as funcionalidades de processamento e otimização de consultas, gerenciamento de transações e o gerenciamento de armazenamento. A máquina Cliente fica com o fornecimento da aplicação, da interface do usuário e um módulo cliente de SGBD responsável pelo gerenciamento dos dados e (algumas vezes) pelo gerenciamento de bloqueios de transações (ÖZSU e VALDURIEZ, 2001).

A comunicação nessa arquitetura é baseada em um protocolo simples, sem conexão, do tipo *Request/Replay(TCP/IP – Transmission Control Protocol / Internet Protocol)*, em que o lado cliente faz uma solicitação(*request*) ao servidor, e este executa todo o processamento necessário e envia a resposta(*replay*) ao cliente.

A manipulação dos dados do banco de dados é feita utilizando uma linguagem de manipulação de dados chamada SQL (*Structured Query Language*). Uma consulta gerada pela SQL em um ambiente Cliente/Servidor pode ser descrita da seguinte maneira:

- O cliente analisa uma consulta gerada pelo usuário, decompõe a consulta global em várias consultas locais, que serão executadas por componentes do SGBD Distribuído em diferentes *hosts* remotos, com a finalidade de localizar os dados nos vários *hosts* independentes.

- Cada consulta local é enviada para o *host* servidor apropriado.

- Cada servidor remoto executa o processamento da consulta local e envia a relação resultante para o *host* cliente.

- O *host* cliente combina os resultados das sub-consultas para produzir o resultado da consulta submetida.

Essa arquitetura é a mais utilizada para o projeto de bancos de dados distribuídos. A arquitetura do sistema é dito homogêneo, pois cada *host* utiliza o mesmo tipo de SGBD e os mesmos esquemas (ÖZSU e VALDURIEZ, 2001) (DATE, 2004).

1.3.3.2 Sistemas distribuídos não-hierárquicos

As características de sistemas distribuídos é a presença de vários computadores geograficamente dispersos, interligados por uma rede de computadores podendo ser de alta velocidade ou de baixa velocidade. Cada computador é fracamente acoplado, pois não há compartilhamento de memória principal como também de disco. E cada computador armazena um fragmento do banco de dados e compartilha este com os demais participantes do sistema (SILBERSCHATZ et al, 1999).

Cada *host* possui a definição de um esquema interno local devido ao armazenamento dos dados ser diferente em cada *host* do sistema.

Para lidar com as formas de distribuição de dados (fragmentação e replicação), surge a necessidade de definição do esquema conceitual local para cada *host*. Os esquemas conceituais local e global como também o mapeamento intermediário proporcionam as transparências de localização e replicação para o sistema. O esquema conceitual global proporciona a transparência de rede, pois esse esquema descreve a estrutura lógica dos dados em todos os *hosts* da rede (ÖZSU e VALDURIEZ, 2001).

A geração de sub-consultas é uma tarefa do esquema conceitual global que converte consultas fornecidas por aplicativos do usuário ou pelos usuários (esquema externo) em sub-consultas (ou consultas locais), que serão mapeadas pelo dicionário/diretório global aos *hosts* que contenham os dados solicitados. Os esquemas conceituais locais se encarregam de executar as operações locais independente de outros *hosts* da rede, com a finalidade de retornar a relação que lhe foi solicitada. O dicionário/diretório local (D/DL) se encarrega de

executar os mapeamentos necessários para obtenção dos dados no Banco de Dados local (ÖZSU e VALDURIEZ, 2001).

Os mapeamentos globais são feitos pelo dicionário/diretório global (D/DG) e os mapeamentos locais são feitos pelo dicionário/diretório local.

A independência de dados nessa arquitetura é garantida, pois é uma extensão do modelo arquitetônico ANSI/SPARC que fornece naturalmente essa característica.

Özsu e Valduriez (2001) definem o Processador do Usuário e o Processador de Dados como sendo os principais componentes de um SGBDD. O Processador do Usuário trata da iteração do sistema com o usuário, e o Processador de Dados trata do armazenamento dos dados. Os componentes de um SGBDD junto com os elementos constituinte dos mesmos podem ser observados na Figura 2.1.

O primeiro componente mostrado pela Figura 2.1 é o Processador do Usuário que consiste em quatro elementos:

1. **Tratador de interface do usuário:** responsável por toda interação com o usuário, interpretação dos comandos submetidos ao SGBD, bem como pela formatação dos resultados que serão fornecidos aos usuários.
2. **Controlador de dados semânticos:** responsável pela autorização ou não pelo processamento da consulta solicitada pelo usuário. Utiliza as restrições de integridade e autorizações definidas no esquema conceitual global.
3. **Otimizador de consultas globais**: responsável pela definição da melhor estratégia para execução das consultas (entre elas junções distribuídas) e conversão das consultas globais em consultas locais, utilizando os esquemas conceitual locais e global como também o diretório global.

4. **Monitor de execução distribuída:** responsável pela coordenação da execução distribuída da solicitação do usuário e pela comunicação com outros monitores localizados em *hosts* remotos.

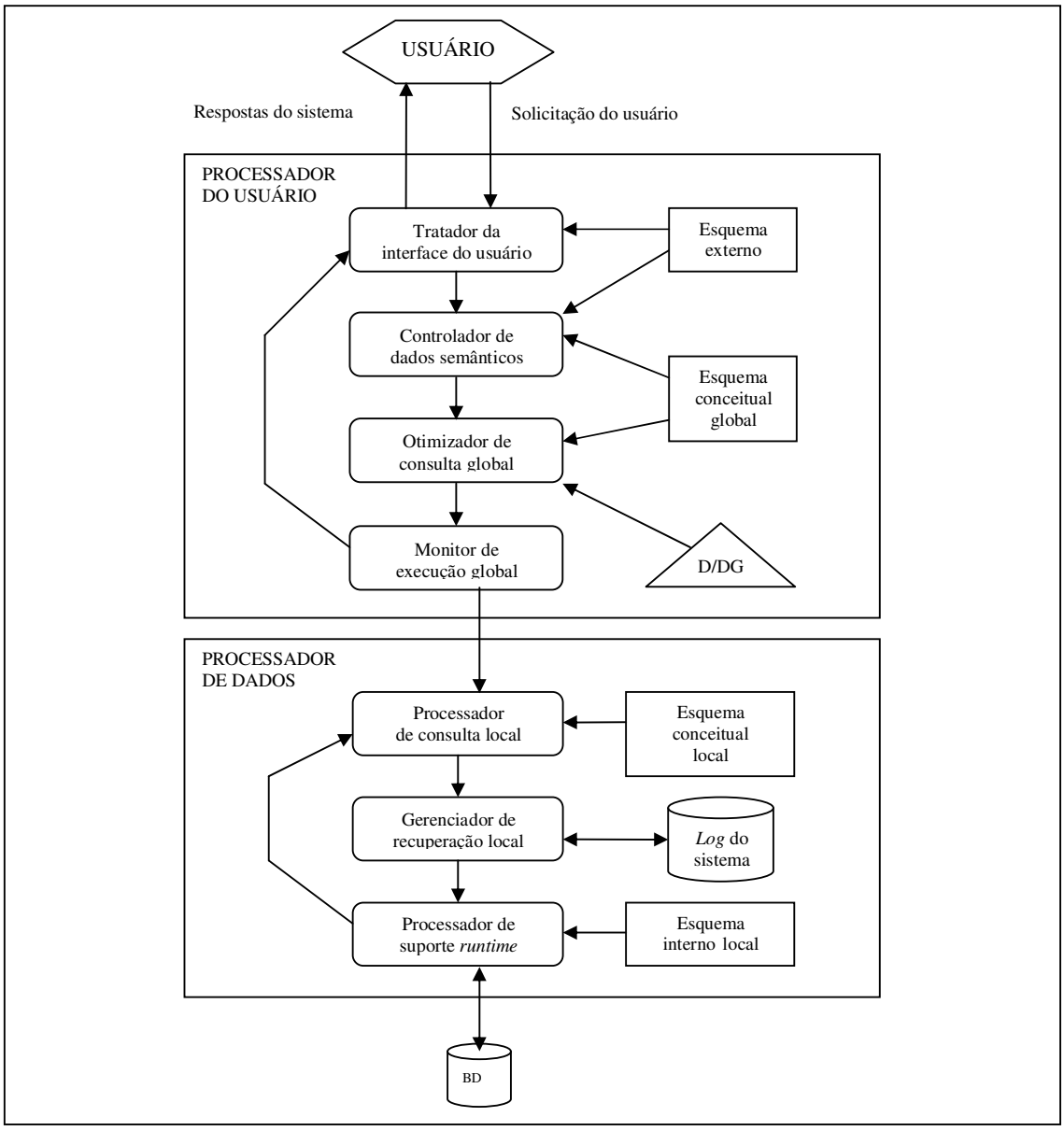


Figura 1.1- Componentes de um SGBD distribuído (ÖZSU e VALDURIEZ, 2001).

O segundo componente é o Processador de Dados que consiste em três elementos:

1. **Otimizador de consulta local:** responsável pela escolha da melhor estratégia de acesso aos dados do banco.
2. **Gerente de recuperação local:** responsável pela consistência do banco de dados, independente de qualquer tipo de falha.
3. **Processador de suporte *runtime*:** responsável pelo acesso físico ao banco de dados, em função das instruções geradas pelo otimizador de consultas e pela gerência dos *buffers* do banco de dados. É a interface para o sistema operacional.

É importante observar que essa arquitetura, diferentemente da arquitetura cliente/servidor, não implica em uma obrigatoriedade de implementação em máquinas diferentes. Assim, *hosts* com a funcionalidade de atualização do Banco de Dados podem ter as duas camadas, enquanto que os *hosts* somente de consultas só necessitam da camada processador do usuário (ÖZSU e VALDURIEZ, 2001).

1.3.3.3 Arquitetura de Sistemas de Vários Bancos de Dados

As arquiteturas de SVBDD e a arquitetura de SBDD são influenciadas pelos diferentes níveis de autonomia e diferenciam-se na definição do esquema conceitual global. Em SGBDDs o esquema conceitual global define a visão conceitual de todo o Banco de Dados. Já no caso de vários SVBDs distribuídos, apenas alguns Bancos de Dados locais podem estar compartilhados. Na Figura 2.2 apresenta-se a arquitetura para vários SGBDs Distribuídos (ÖZSU e VALDURIEZ, 2001).

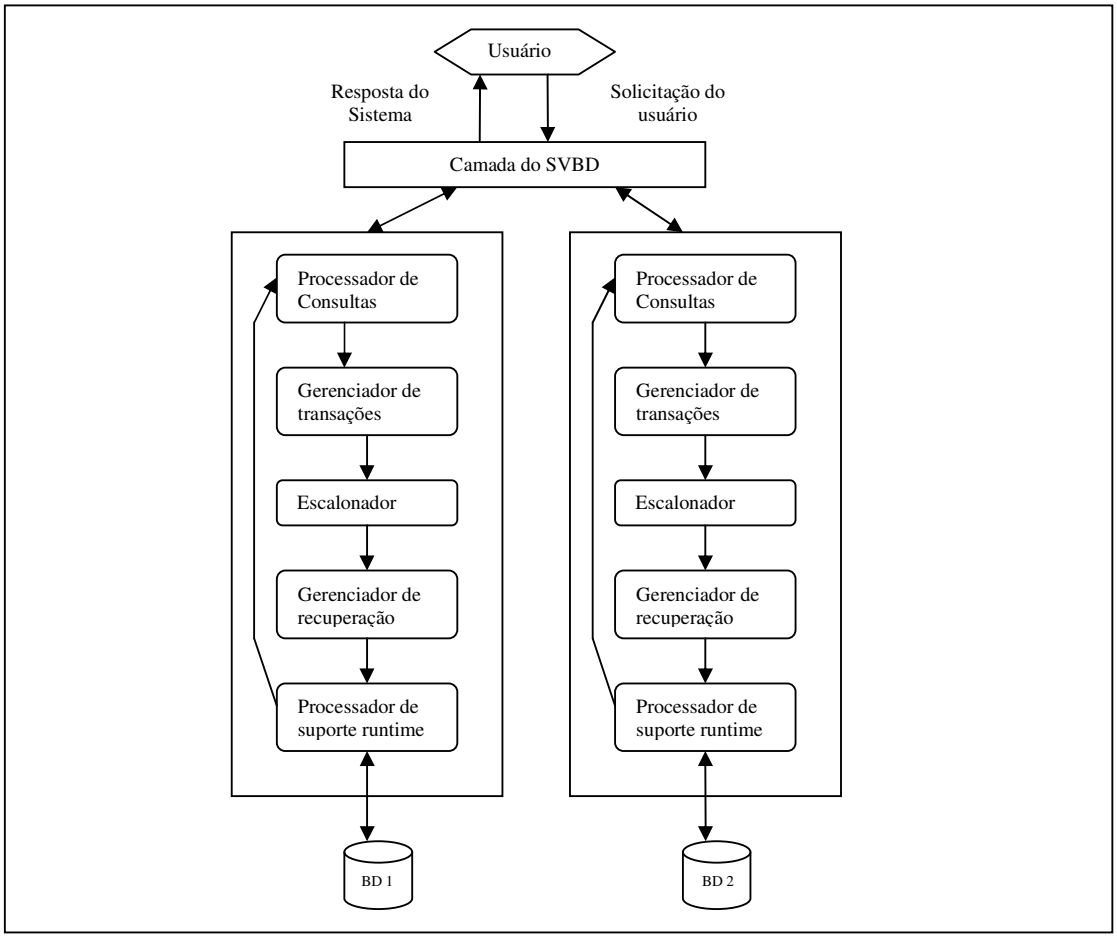


Figura 1.2 - Componentes de um SVBD (ÖZSU e VALDURIEZ, 2001).

Esta arquitetura é significantemente diferente das arquiteturas anteriores, pois nela existem SGBDs completos, cada um dos quais responsável pela gerência de um Banco de Dados distinto. Esta arquitetura fornece uma camada de software denominada de *multi-SGBD* que funciona coordenando a integração entre os diferentes SGBDs, fornecendo todos os recursos necessários para acesso aos diversos Bancos de Dados. Este software é apresentado na Figura 1.2 como sendo a Camada do SVBD.

1.4 Formas de distribuição de dados

As relações (tabelas) existentes em um Banco de Dados Distribuído são divididas de forma apropriada em sub-relações ou unidades lógicas, chamadas de fragmentos. Cada *host* poderá ser apontado como sendo o local para o armazenamento de um fragmento.

Segundo Date (2004) os dados em um sistema distribuído, de acordo com seu armazenamento físico, isto é, sua alocação, podem ser particionados ou replicados. Portanto, deve-se esclarecer que o particionamento é a alocação dos fragmentos de forma não-redundante e a replicação é a alocação de um mesmo fragmento em mais de um local, isto é, de forma redundante.

As técnicas de fragmentação, particionamento, replicação e alocação são tratadas no projeto de distribuição de dados.

1.4.1 Fragmentação de dados

No processo de distribuição de dados, as unidades lógicas do banco de dados deverão ser identificadas.

Segundo Elmasri e Navathe (2005) a unidade lógica mais simples é a relação, podendo esta ser dividida em unidades lógicas menores para a distribuição entre os *hosts*. Estas unidades lógicas, como citadas anteriormente, são os fragmentos.

Existem três tipos fundamentais de fragmentação:

- **Fragmentação horizontal:** divide a relação com base em suas tuplas (linhas), ou seja, os fragmentos resultantes conterão um subconjunto das tuplas da relação original que satisfazem uma condição de seleção sobre os atributos da relação. Há duas versões da fragmentação horizontal: primária e derivada. A *fragmentação horizontal primária* consiste na divisão da relação

com base em predicados definidos sobre a própria relação. Já a *fragmentação derivada* é a divisão da relação de acordo com os predicados envolvendo relações dominantes e outras relações.

- **Fragmentação vertical:** consiste no particionamento da relação com base em seus atributos. O resultado da fragmentação vertical são sub-relações constituída por um conjunto de atributos selecionados através de projeções sobre a relação original. Portanto, dada uma relação R a fragmentação irá produzir fragmentos R_1, R_2,..., R_r, onde cada fragmento é uma relação de menor tamanho que possui um sub-conjunto dos atributos da relação R como também a chave primária de R. Freqüentemente é necessário adicionar um atributo especial e exclusivo, chamado *identificadores de tuplas* ou *tupla_id* no esquema R. O principal objetivo da fragmentação vertical é dividir uma relação em pequenas relações, agrupando determinados atributos mais solicitados pelos aplicativos do usuário. Em termos gerais, uma fragmentação ótima é aquela que produz um esquema de fragmentação no qual o aplicativo do usuário acessa apenas um fragmento, resultando na minimização do tempo de execução dos aplicativos dos usuários.
- **Fragmentação mista ou híbrida:** Consiste na aplicação das duas estratégias de fragmentação citadas anteriormente, executadas uma após a outra sobre uma relação. O resultado é um conjunto de fragmentos divididos pela fragmentação horizontal, logo em seguida cada um destes fragmentos é particionado em mais dois fragmentos verticais ou vice-versa. A reconstrução da relação original é feita através de junção natural nos fragmentos verticais, com operações de união aplicadas nos fragmentos horizontais, na sequência.

Para exemplificar as formas de fragmentação pode-se tomar como base uma relação de contas de um banco nacional, apresentada pela Figura 2.3.

nome_agência	número_conta	nome_cliente	saldo
Marília	320-1238	Carlos	1200,00
Marília	320-4569	Ana Maria	5400,00
Porto Alegre	110-4213	Juca	2500,00
Brasília	250-5489	João	1500,00
Brasília	250-0147	Cristiane	5200,00
Porto Alegre	110-5236	Elaine	10800,00

Figura 1.3 - Relação depósito.

Na Figura 2.3, o atributo *nome_agência* possui instâncias de agências localizadas em Marília, Porto Alegre e Brasília, que será usada como base para a fragmentação. Porém, a agência localizada em Marília não necessariamente precisa ter informações de contas localizadas em Porto Alegre ou Brasília. O particionamento da relação será de modo que as tuplas que possuem atributo base idêntico serão armazenadas em relações diferentes e cada relação será armazenada em um *host* diferente. Na Figura 2.4 exibe-se o resultado da fragmentação horizontal.

nome_agência	número_conta	nome_cliente	saldo
Marília	320-1238	Carlos	1200,00
Marília	320-4569	Ana Maria	5400,00

depósito1

nome_agência	número_conta	nome_cliente	saldo
Porto Alegre	110-4213	Juca	2500,00
Porto Alegre	110-5236	Elaine	10800,00

depósito2

nome_agência	número_conta	nome_cliente	saldo
Brasília	250-5489	João	1500,00
Brasília	250-0147	Cristiane	5200,00

depósito3

Figura 1.4 - Fragmentos horizontais da relação depósito.

O fragmento depósito1 será armazenado no *host* de Marília, o *host* de Porto Alegre armazenará o fragmento depósito2 e o fragmento depósito3 será armazenado no *host* de Brasília. A fragmentação horizontal apresentada na Figura 2.4 é considerada completa devido ao fato de que todas as tuplas da relação depósito aparecem em algum dos fragmentos. Nesse caso, a fragmentação completa é também disjunta, pois as tuplas da relação original não satisfazem as duas condições impostas para gerar os fragmentos.

Considerando que os atributos mais solicitados por um *host* A são os atributos referentes à conta e um *host* B acessa freqüentemente os atributos referentes aos clientes, a fragmentação vertical divide a relação depósito em duas sub-relações de modo que o *host* A conterá somente fragmento depósito1, enquanto o *host* B armazenará somente o fragmento depósito2.

Como mostrado na Figura 2.5, na maioria dos casos, há necessidade de acrescentar um atributo exclusivo à relação original. Executando a junção natural dos fragmentos, originados pela fragmentação vertical, com o atributo de junção *tupla_id* é possível obter a relação depósito.

nome_agência	nome_cliente	*tupla_id*
Marília	Carlos	1
Marília	Ana Maria	2
Porto Alegre	Juca	3
Brasília	João	4
Brasília	Cristiane	5
Porto Alegre	Elaine	6

depósito 1

número_conta	saldo	*tupla_id*
320-1238	1200,00	1
320-4569	5400,00	2
110-4213	2500,00	3
250-5489	1500,00	4
250-0147	5200,00	5
110-5236	10800,00	6

depósito2

Figura 1.5 - Fragmentos verticais da relação depósito.

Aplicando a fragmentação vertical em cada fragmento apresentado na Figura 2.4, é possível obter a fragmentação mista com seis fragmentos. O resultado dessa fragmentação são fragmentos depósito i e j, onde i representa o número de fragmentos horizontais (1, 2 e 3) e j representa o número de fragmentos verticais (1 e 2). Os fragmentos apresentarão os seguintes esquemas:

Esquema_depósito 1,j(nome_agência, nome_cliente, tupla_id)

Esquema_depósito i,2(número_conta, saldo, tupla_id).

Portanto a fragmentação mista, como dito anteriormente, é a aplicação das duas técnicas, uma após a outra, obtendo, assim, fragmentos horizontais particionados verticalmente.

1.4.2 Alocação dos dados

A alocação trata do problema de armazenamento dos fragmentos, ou cópias de fragmentos, pelos diversos *hosts* da rede (distribuição de dados) levando em consideração o caráter ótimo de custos e de desempenho em que os fragmentos serão armazenados.

As funções de custo de armazenamento dos fragmentos pelos *hosts*, como de operações executadas sobre esses fragmentos e da comunicação de dados, são métricas que determinarão o local adequado para alocar um determinado fragmento.

Deve-se proporcionar a menor perda do desempenho, ou seja, manter mínimo o custo de degradação do desempenho do sistema como também minimizar o tempo de resposta para o usuário.

Portanto, a alocação deve oferecer um esquema que minimize o tempo de resposta das consultas do usuário, enquanto o custo de processamento é mínimo.

1.4.3 Replicação dos dados

A replicação é executada por razão de confiabilidade, disponibilidade e eficiência de transações de somente leitura. A decisão para replicar um banco de dados parte da análise de consultas de leitura e consultas de atualização (SILBERSCHATZ et al, 1999) (ELMASRI e NAVATHE, 2005).

O caso mais extremo é a existência de uma cópia por inteiro do Banco de Dados armazenado em cada *host*, criando um banco de dados distribuído completamente replicado para cada *host*. Essa estratégia pode proporcionar a melhoria na disponibilidade, porque o sistema pode manter-se em operação, desde que pelo menos um *host* esteja funcionando e também a melhoria no desempenho na recuperação de dados globais, devido à consulta ser gerada e executada no *host* local, independente do *host*. A desvantagem está intimamente relacionada à consistência dos dados, visto que a replicação de todos os dados pode reduzir drasticamente a velocidade nas operações de atualização, devido à execução de uma única atualização lógica por todos os *hosts* que contenham uma réplica do banco de dados.

No outro extremo da replicação está a não replicação de um Banco de Dados, chamado Banco de Dados particionado, que implica na alocação não redundante de diferentes fragmentos em *hosts* distintos da rede.

A replicação parcial dos dados surge entre esses dois extremos, apresentados anteriormente, proporcionando a replicação de determinados fragmentos do Banco de Dados, enquanto outros fragmentos não serão replicados. O número de cópias de cada fragmento deve estar entre um intervalo de no mínimo um e no máximo o número de *hosts* participantes do sistema. Um caso especial de replicação parcial está acontecendo com maior freqüência em aplicações que envolvem computação móvel, pois o Banco de Dados é parcialmente replicado para computadores móveis que possuem aplicativos que se encarregam de sincronizar periodicamente com o Banco de Dados móvel com o servidor de Banco de Dados.

A descrição dos fragmentos de replicação são chamados de esquema de replicação (ELMASRI e NAVATHE, 2005).

Como no processo de alocação dos dados, o grau de replicação depende do desempenho e disponibilidade desejados para o sistema como também da freqüência de acesso de transações submetidas em cada *host*. Por exemplo: se for necessário alta disponibilidade, se as transações podem ser submetidas em qualquer *host*, e a maior parte das transações é somente leitura, então um Banco de Dados totalmente replicado é uma boa escolha.

Porém, se certas transações acessam partes específicas do banco de dados a partir de um *host* particular, então o conjunto de fragmentos correspondentes precisam ser alocados neste *host*. Os dados que são necessários para diversos *hosts*, podem ser replicados para todos eles. Se muitas atualizações são executadas, então é melhor limitar o uso da replicação.

Portanto se o cálculo da razão de consultas de somente leitura e consultas de atualizações resultar em um valor igual ou maior que 1 então a replicação se torna vantajosa, caso contrário a replicação poderá causar sérios problemas (ELMASRI e NAVATHE, 2005).

As informações referentes a fragmentação, alocação e replicação de dados são armazenadas no catálogo/diretório global, que é acessado pelos aplicativos do SBDD quando necessita-se de alguma informação.

2 - ARMAZENAMENTO E RECUPERAÇÃO DE IMAGENS MÉDICAS

As imagens geradas por equipamentos digitais de diagnóstico são algumas das ferramentas mais importantes na prática da Medicina, sendo amplamente usadas em hospitais, clínicas e centros médicos. Auxiliam os profissionais da saúde em diagnóstico tratamentos e planejamento de cirurgias, além de fornecer uma visualização de órgãos internos, tecidos, ossos e outras estruturas do corpo do paciente.

Os centros de diagnósticos por imagens geram grande quantidade de imagens e informações relacionadas aos pacientes. Com isso o gerenciamento (armazenamento, localização e distribuição) das imagens vem se tornando um ponto crítico nos ambientes clínicos. Para solucionar o problema, Sistemas de Comunicação e Armazenamento de Imagens (PACS - *Picture Archiving and Communication System*), Sistemas de Informação de Radiologia (RIS – *Radiology Information System*) e Sistemas de Informações Hospitalares (HIS – *Hospital Information System*) estão sendo integrados com os sistemas já existentes em hospitais e clínicas. Tais sistemas permitem o gerenciamento do grande volume de dados gerados por diferentes sistemas, acesso às informações do paciente de forma rápida e distribuída, melhorando, assim, a qualidade e agilidade dos serviços prestados (KOBAYASHI, 2002) (SANTOS e RUIZ, 2002).

2.1 PACS – *Picture Archiving and Communication System*

PACS são sistemas de gerenciamento clínico/hospitalar de ampla aceitação mundial que possibilita o armazenamento, a recuperação, a transmissão e a visualização de grandes volumes de dados. Estes dados podem estar localizados na máquina onde o armazenamento/recuperação está sendo realizado, ou em locais físicos distintos.

Um dos objetivos desse tipo de sistema é proporcionar a integração e a comunicação entre os diversos tipos de equipamentos médicos que trabalham com imagens. Esforços estão sendo feitos para a descentralização da informação, em particular as imagens médicas, permitindo que as informações sejam acessadas não apenas onde foi gerada, como também em locais físicos distintos (NETO et al, 2005). Isso gera uma melhor disponibilidade dos dados oferecidos pelo sistema de Banco de Dados distribuído em conjunto com o sistema PACS.

Ao mesmo tempo, tais meios devem prover uma qualidade de serviço suficiente, de forma que a troca dos dados entre os diferentes setores possa ser feita de uma forma rápida, confiável e eficiente (KOBAYASHI, 2002).

Esses sistemas permitem a integração rápida de usuários e médicos com os diversos setores da unidade de saúde como TC – Tomografia Computadorizada, RS – Ressonância Magnética, US – Ultra-som, RX – Raio X e MD – Microscopia digital, entre outros. Na tentativa de adquirir informações que não estejam armazenadas no setor local, os usuários desses sistemas não necessitam deslocar-se até outro setor para a obtenção da informação, bastando apenas solicitar ao sistema local as informações necessárias. O sistema cuida do processamento, localização e exibição das mesmas.

Segundo Santos e Ruiz (2002), os sistemas PACS podem apresentar alguns fatores que os tornam inviável, fatores como custos elevados de desenvolvimento, adequação do funcionamento do sistema às necessidades do serviço, ajuste de estruturas de rede de computadores, instalação e manutenção dos equipamentos e integração dos sistemas já existentes.

Alguns padrões devem ser considerados no desenvolvimento de sistemas PACS, entre eles, TCP/IP, DICOM e HL7.

2.1.1 TCP/IP (*Transmission Control Protocol / Internet Protocol*)

O protocolo mais utilizado para a transmissão de dados na área médica é o protocolo TCP/IP. Os dados que trafegam nos PACs são informações referentes ao paciente e como também as suas respectivas imagens médicas.

As sete camadas do protocolo *Open Standards Interconnect* (OSI) da *International Standards Organization* (ISO) (aplicação, apresentação, sessão, transporte, rede, enlace, e por fim, a camada física) definem como deve ocorrer a transmissão da informação desde os bits que trafegam pelas fibras e cabos metálicos, até a mensagem a ser exibida para o usuário.

2.1.2 DICOM (*Digital Imaging and Communication in Medicene*)

O DICOM é um padrão desenvolvido por um comitê de trabalho formado por membros do *American College of Radiology* (ACR) e do *National Electrical Manufactures Association* (NEMA), que iniciou os trabalhos em 1983 com o intuito de desenvolver um padrão digital de informações e imagens. O comitê publicou a primeira versão em 1985, que foi chamada de ACR-NEMA 300-1985 ou (ACR-NEMA Version 1.0) e a segunda versão em 1988, chamada de ACR-NEMA 300-1988 ou (ACR-NEMA Version 2.0) (MEDICAL IMAGE, 2005).

A terceira versão do padrão chamado DICOM 3.0 foi apresentado em 1993 com o intuito de proporcionar a comunicação entre os equipamentos de diferentes fabricantes, facilitar o desenvolvimento e expansão dos sistemas PACS e permitir a criação de uma base de dados de informações de diagnósticos que possam ser examinadas por uma grande variedade de aparelhos distribuídos fisicamente em entidades de saúde (CLUNIE, 2005).

O padrão DICOM especifica um protocolo de comunicação baseado em TCP/IP, define a operação de serviços e cria um mecanismo para identificação única de objetos. Este

padrão garante que uma grande variedade de equipamentos seja interconectados e que os dados possam ser reconhecidos e interpretados corretamente por sistemas produzidos por diferentes fabricantes. As especificações DICOM continuam sua evolução por meio de requisições e discussões da comunidade da ACR-NEMA. No ano de 2000 foi aprovado mais um suplemento DICOM denominado *Structured Reporting* (DICOM-SR) para o desenvolvimento de documentos estruturados que representam a informação médica relacionada a uma imagem (CLUNIE, 2005).

Segundo Moreno (2005), a utilização de equipamentos que suportam DICOM em hospitais e instituições de pesquisa do Brasil é significativamente grande. Praticamente todas as instituições da saúde que trabalham com equipamentos que utilizam imagem digital realizam algum tipo de gerenciamento DICOM.

2.1.3 HL7 (*Health Level 7*)

Para a comunicação de informação médica tornou-se relevante o uso do padrão HL7 (DEVAKI et al, 2001), em desenvolvimento desde 1997 e sancionado pelo *American National Standards Institute* (ANSI), para trocar dados clínicos de pacientes entre laboratórios. O objetivo do padrão HL7 é prover padrões para o intercâmbio, gerenciamento e integração de dados que suportam o atendimento clínico de pacientes e o gerenciamento, fornecimento e avaliação de serviços de atendimento à saúde. Mais especificamente, o HL7 propõe-se a criar abordagens flexíveis e de baixo custo, padrões, metodologias e serviços relacionados para estimular a interoperabilidade entre sistemas de informação em saúde.

O padrão HL7 tem sido utilizado para facilitar a troca de informações entre sistemas baseados em DICOM em algumas aplicações, porém, ainda há questões não completamente solucionadas em relação à integração desses padrões (CLUNIE) (DEVAKI et al, 2001).

2.2 Formatos de Imagens Digitais

Existem basicamente dois tipos de padrões para armazenamento de imagem: *bitmap* e vetorial. O formato *bitmap* utiliza mapas de *bits*, em que cada ponto da imagem é armazenado em uma matriz e cada ponto possui um valor particular de luminosidade e cor. Esse formato pode utilizar sistema de compressão de arquivos com perda (*lossy*) ou sem perda (*loss-less*). Os formatos vetoriais trabalham sobre uma matriz semelhante ao mapa de *bits* das imagens *bitmap*, porém as imagens vetoriais não guardam os pontos da imagem, mas sim a localização dos pontos na matriz e outros atributos como: coordenadas x e y, cor, transparência, estilo. Desta forma, a imagem ao ser redimensionada não tem sua qualidade afetada. O tamanho dos arquivos gerados nesse formato é muito menor que os arquivos gerados no formato *bitmap*, devido à utilização de informações na forma geométrica (COELHO, 2004).

Os formatos de imagens mais utilizados atualmente são os formatos JPEG, GIF, PNG, e TIFF, que têm como base o padrão de imagens *bitmap*.

2.2.1 GIF (*Graphics Interchange Format*)

O formato GIF, criado pela Compuserve em 1987 (KELNER e FRERY, 2004) juntamente com o formato JPEG, têm alcançado uma grande popularidade por serem ambos os mais utilizados na Internet. As imagens GIF utilizam uma tabela de cores para representar a paleta indexada de 8 *bits* por *pixel,* limitando-se a armazenar no máximo 256 cores. Esse formato utiliza um algoritmo de compressão sem perda (LZW). O formato GIF é indicado para situações nas quais uma parte da imagem é transparente ou contém pequenas animações. Para fotografia digital são indicados outros formatos como JPEG e TIFF (MARIANO, 2004).

O algoritmo compressão LZW, utilizado por esse formato, é um algoritmo proprietário da Unisys que proporciona a gravação e regravação da imagem sem ocorrer

degradação da qualidade fazendo com que novas cópias sejam idênticas à primeira. Esse formato possui algumas vantagens como:

- Suporta animações.
- Preserva a qualidade original da foto.
- Seu tamanho em relação aos outros padrões é relativamente pequeno.

Essas vantagens tornam o formato GIF o formato ideal para trabalhar com imagens e animações na Web.

O GIF está disponível em duas versões: a primeira versão do GIF surgiu em 1987 (GIF87a) e em 1989 foi disponibilizada uma nova versão a GIF89a, que suportava animações, transparência e entrelaçamento (imagem é descarregada aos poucos).

2.2.2 JPEG (*Join Photographic Experts Group*)

O formato JPEG ou JPG é um dos mais utilizados no cotidiano. Possui diferentes níveis de compressão com perda de qualidade e sua principal vantagem é permitir a redução do tamanho das imagens de uma maneira muito significativa. Permite qualquer resolução espacial e qualquer profundidade de *bits*, sendo o mais apropriado para fotografia digital.

De acordo com Kelner e Frery (2004), o formato JPEG possui as seguintes características:

- Utiliza um algoritmo de compressão com perda, mas o grau de compressão pode ser ajustado.
- Suporta 24 *bits* de cores seguindo o modelo RGB.
- A cada vez que a imagem nesse formato é salva ela sofre uma degradação de sua qualidade.

O algoritmo utilizado para a compressão trabalha desprezando os componentes de baixa e alta freqüência, resultando em uma imagem que o olho humano não percebe as alterações.

2.2.3 PNG (*Portable Network Graphics*)

Este formato representa uma tentativa de padronizar os formatos gráficos presentes na Internet combinando as possibilidades de transparência e animação do GIF com as possibilidades de compressão, resolução e cores de JPEG. Essas características atribuídas ao formato PNG buscam o balanceamento da qualidade e da redução do tamanho do arquivo (MARIANO, 2005).

O formato PNG é bastante recente e foi desenvolvido com o objetivo de ser um padrão aberto de arquivo para servir de alternativa ao formato GIF, que é patente da Compuserve, por isso manteve a mesma estratégia de compressão sem perdas. O formato PNG utiliza os algoritmos de compressão Lempel-Ziv 77 (LZ77) e Huffman (COELHO, 2004).

Segundo Coelho (2004), o formato PNG apresenta as seguintes vantagens:

- Possibilita a utilização de um número maior de cores que os formatos GIF e JPEG; 48 bits de cores por *pixel* para *truecolor* (24 bits do RGB, 8 bits do canal *Alpha* e 16 bits para escalas de cinza), ou 16 bits por *pixel* para escalas de cinza.
- Devido à utilização do algoritmo LZ77, possibilita um grau de compressão de até 30% maior dos arquivos com menor perda da qualidade em relação aos formatos GIF e JPEG.

2.2.4 TIFF (*Tagged Image File Format*)

É um formato de arquivo aceito por praticamente todos os programas de manipulação de imagem. Foi desenvolvido em 1986 pela Aldus e pela Microsoft numa tentativa de criar um padrão para imagens geradas por equipamentos digitais (MARIANO, 2005).

O TIFF é capaz de armazenar imagens *truecolor* (24 ou 32 bits) permitindo ao arquivo ter alta ou baixa qualidade. Algumas vezes se utiliza um algoritmo de compressão LZW, mas o ideal é a geração de arquivos TIFF sem compressão para se escapar de possíveis problemas de compatibilidade entre sistemas. Esse formato de arquivo não apresenta limitações na resolução espacial e nem na profundidade de *bits* (GRAPHICS,2005).

Algumas vantagens oferecidas por esse formato de arquivo são:

- Sua estrutura é suportada por diversas aplicações.
- Independe da arquitetura de um computador, sistemas operacionais e plataformas gráficas.
- Podem ser ajustados às características de um *scanner*, monitor ou impressora.
- Permite salvar campos informativos dentro do arquivo.

Devido às vantagens que este formato apresenta, ele é muito utilizado na representação de imagens digitalizadas a partir de *scanners laser*, que são utilizados para capturar imagens médicas (NUNES et al, 2001).

Na Figura 3.1 é apresentada a característica de cada formato apresentado nessa Seção.

Formatos Bitmap				
Atributos/Formatos	GIF	JPEG	PNG	TIFF
Dimensão do arquivo gerado	Pequeno	Pequeno	Pequeno	Grande
Algoritmo de compressão	Sem perda	Com perda	Sem perda	Sem perda ou não utiliza
Quantidade de *bits*	8 *bits* (256 cores)	24 *bits* de cores	48 *bits* de cores	24 ou 32 *bits* de cores
Indicado para:	*WEB*, Imagens com transparência e pequenas animações	Fotografia digital e Páginas da *WEB*	*WEB*	Equipamentos que necessitam de imagens de alta qualidade como *scanners laser*

Figura 2.1 - Comparação entre os formatos de imagem digital

3 - SGBD ORACLE9i DISTRIBUÍDO

Como citado no Capítulo 1, em um sistema de Banco de Dados distribuído cada máquina participante do sistema pode acessar informações que residem em seu próprio banco de dados como também informações residentes em outras máquinas participantes do sistema. Esses dados parecem estar armazenados logicamente em um computador, ou seja, um usuário que está conectado a um Banco de Dados "A" executa uma instrução SQL simples que para o mesmo parece acessar o Banco de Dados local, mas não necessariamente a informação pode estar armazenada no Banco de Dados "A", podendo acessar um Banco de Dados "B" para obter a informação. Isso faz com que o usuário não tenha conhecimento da existência da rede e da distribuição dos dados.

3.1 Arquiteturas de Banco de dados Oracle

Para implementação de sistemas distribuídos, o Oracle disponibiliza dois tipos de arquiteturas classificadas quanto aos tipos dos Bancos de Dados que compõem o sistema:

- Sistema de Banco de Dados Distribuído Homogêneo: Esse tipo de sistema é composto por apenas bancos de dados Oracle, podendo ser ou não da mesma versão.
- Sistema de Banco de Dados Distribuído Heterogêneo: pelo menos um dos bancos de dados que compõe o sistema é de outro fabricante.

A arquitetura de sistema distribuído Oracle utiliza um ambiente de rede integrado com as arquiteturas cliente-servidor e Banco de Dados Distribuído.

3.1.1 Cliente/Servidor no Oracle

Na arquitetura cliente-servidor, o SGBD Oracle é dividido em duas partes: a parte do cliente que é também chamada de *front-end* e o *back-end,* que é a parte do servidor (ELMASRI e NAVATHE, 2005) (ORACLE, 2005).

Nessa arquitetura o processo cliente solicita serviços ao processo servidor, o processo servidor trata as solicitações, processa e retorna o resultado ao cliente. Trata também das funções relacionadas com o acesso compartilhado concorrente. Cada *host* é um servidor de Banco de Dados Oracle que pode atuar como cliente ou servidor (arquitetura servidor/servidor). Um *host* é dito ser cliente quando o mesmo solicita serviços de um servidor Oracle.

A Figura 3.1 apresenta a arquitetura cliente/servidor do Oracle, na qual um processo cliente, localizado na máquina que contém o Banco de Dados DB1, solicita serviço ao processo servidor, localizado na máquina que contém o banco de dados DB2, utilizando o Oracle Net que é o software responsável por estabelecer e manter a conexão entre processos clientes/servidores Oracle. Um *Link* de Banco de Dados é necessário para que o processo cliente possa acessar informações na máquina na qual o processo servidor reside.

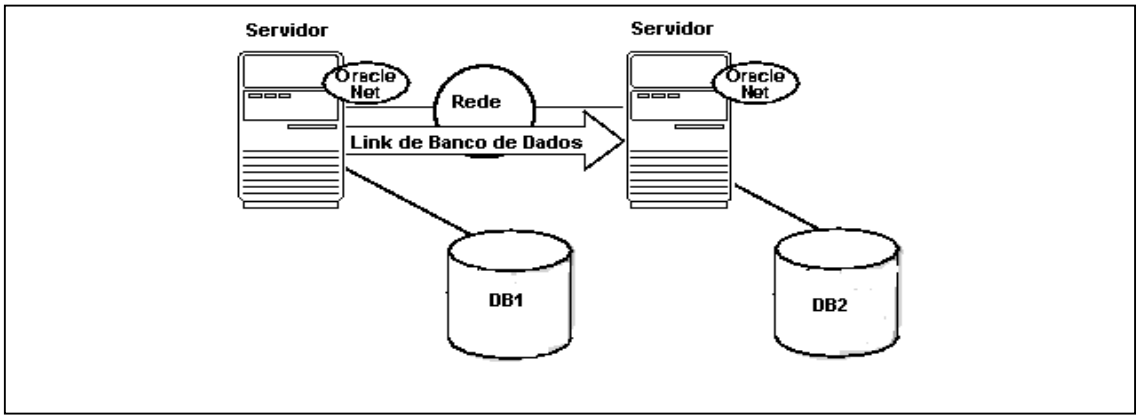


Figura 3.1- Arquitetura Cliente/Servidor Oracle (ORACLE, 2005)

Para que haja a comunicação entre o cliente e o servidor, os mesmos devem utilizar o mesmo protocolo de comunicação.

A conexão entre o cliente e servidor pode ocorrer de duas formas:

- Direta: usuário conecta a um banco de dados servidor e executa instruções SQL que acessam a base de dados a qual o cliente conectou.
- Indireta: usuário conecta em um banco de dados servidor e acessa informações de um banco de dados remoto.

3.1.2 Sistema de Banco de Dados Distribuído Homogêneo

Um Sistema de Banco de Dados Distribuído Homegêneo é uma rede de dois ou mais Bancos de Dados Oracle que residem em uma ou mais maquinas participantes da rede. Nesse tipo de sistema uma aplicação do usuário pode acessar ou modificar diversas informações contidas em diferentes Bancos de Dados.

Para a aplicação do cliente, a existência da rede e da distribuição de dados deve ser transparente. O Oracle oferece sinônimos (*synonyms*), visões (*view*) e procedimentos (*procedures*) que possibilitam ao usuário acessar Bancos de Dados remotos com a mesma sintaxe que acessariam um Banco de Dados local.

3.1.3 Sistema de Banco de Dados Distribuído Heterogêneo

Nessa arquitetura de Banco de Dados Distribuído, ao menos um Banco de Dados não é Oracle. O acesso a Bancos de Dados não Oracle é fornecido pelos serviços heterogêneos (*Oracle Heterogeneous Services*) em conjunto com um agente (*agent*). Os serviços heterogêneos permitem o acesso de dados e serviços originários de um Banco de Dados Oracle, utilizando a conectividade genérica via ODBC e OLE-DB inclusos com o Banco de

Dados. Se a conectividade genérica não for utilizada, cada Banco de Dados do sistema heterogêneo deverá prover um *Transparent Gateway* específico do Banco de Dados. (ORACLE, 2005).

3.2 Replicação de dados no Oracle

Como visto na Seção 2.4.3, Capítulo 2, a replicação pode ser utilizada para diminuir o tráfego da rede e melhorar a disponibilidade dos dados. Esses fatores podem ser alcançados devido à existência de réplicas pelos diversos *hosts* da rede. Uma aplicação do Banco de Dados pode solicitar informações de um tabela remota, mas se existe uma réplica da tabela remota no *host* local os dados retornados por uma consulta SQL serão os dados da réplica, e não será necessária a utilização da rede para a obtenção dos dados na tabela remota.

O SGBD Oracle disponibiliza dois tipos de replicação de dados, a replicação básica, também conhecida como *snapshots* (instantâneos) e a replicação avançada ou tabelas mestras replicadas (ORACLE,2005) (ELMASRI e NAVATHE, 2004).

- Replicação Básica: as réplicas das tabelas permitem acesso somente de leitura. Atualizações são feitas somente na tabela mestra (da qual originou os dados) e essas atualizações são propagadas para as réplicas.
- Replicação Avançada: estende-se para além da replicação básica permitindo o acesso de leitura e escrita, por qualquer *host*, nas réplicas existentes no SGBDD replicado. O *Oracle Database Server* é o responsável em garantir a integridade e consistência dos dados.

A replicação básica fornece apenas a replicação dos dados, já a replicação avançada fornece a replicação de outros objetos do Banco de Dados como índices e procedimentos (ORACLE, 2005).

As atualizações dos dados nas réplicas das tabelas mestras são feitas em lote, tornando os dados das replicas consistentes com os dados da tabela mestra. O Oracle deve periodicamente atualizar os *snapshots* com base no período de tempo informado pelo criador do *snapshot*. Os *snapshots* são organizados e agrupados como um grupo de atualização (*refresh)* no qual o Oracle executa a atualização em todos os *snapchots* como uma simples operação.

3.3 Oracle Net Services

Os Serviços de Rede do Oracle (*Oracle Net Services*) são responsáveis em prover a conectividade para sistemas computacionais distribuídos e heterogêneos. Simplificam as complexidades da configuração e da gerência de rede, maximizam o desempenho, e provêem melhorias no diagnóstico de rede (ORACLE, 2005).

A transparência de localização é fornecida pelos Serviços de Rede do Oracle. Por exemplo: considerando que uma empresa possui três Bancos de Dados Recursos Humanos, Marketing e Financeiro, cada banco de dados pode possuir um ou mais serviços associados a ele. Cada serviço é identificado por um nome de serviço (*service name*). Se um usuário acessa o Banco de Dados Recursos Humanos e deseja obter informações do Banco de Dados Financeiro, que é remoto e é conhecido pelo nome de serviço *financeiro*, o usuário deverá apenas fornecer esse nome de serviço para ter acesso à Base de Dados financeiro. Com isso, o usuário não necessita saber da localização do Banco de Dados.

As informações referentes aos serviços existentes são armazenadas em um repositório local (*tnsnames.ora*). O repositório é representado por um ou mais métodos de nomeação. Os Serviços de Rede do Oracle disponibilizam diversos tipos de métodos de

nomeação que suportam a configuração localizada em cada cliente ou uma configuração centralizada que pode ser acessada por todos os clientes da rede (ORACLE, 2005).

O Serviço de Rede do Oracle é baseado no modelo OSI, citado na Seção 3.1.1, Capítulo 3, oferecendo suporte aos protocolos de rede TCP/IP, IM LU6.2 e DECnet.

O pacote *Oracle Net Services* é composto dos seguintes softwares:

- *Oracle Net*
- *Listener*
- *Oracle Connection Manager*
- *Oracle Net Configuration Assistant*
- *Oracle Net Manager.*

3.3.1 *Oracle Net*

O *Oracle Net*, ou Net8 em versões anteriores do Oracle, é responsável por estabelecer e manter a conexão de um cliente para um servidor de banco de dados, ou de um servidor para um outro servidor de banco de dados. Ele atua como um mensageiro entre o cliente e o servidor, sendo o responsável pela a troca de informação entre as duas partes. Este *software* deve estar presente tanto no cliente como também no servidor para que possa ocorrer a comunicação entre os bancos de dados.

O *Oracle Net* é o responsável em fornecer a transparência de rede, pois é ele quem realiza as comunicações necessárias entre as duas partes (ORACLE, 2005).

3.3.2 *Listener*

O *Listener* (ouvinte) é um processo que espera por conexões do cliente para acessar uma base de dados ou diversas bases de dados. Algumas características são citadas abaixo.

- Um processo *Listener* pode atender conexões para acessar Bancos de Dados diferentes.
- Pode atender conexões com diferentes protocolos.
- O nome do *Listener* deve ser único.

Um banco de dados pode operar como servidor dedicado ou como servidor compartilhado. No modo servidor dedicado, o *Listener* passará a requisição do cliente diretamente a um processo servidor dedicado para atender a solicitação do usuário. Esse tipo de processo limita a escalabilidade de usuários no sistema.

No modo servidor compartilhado as requisições feitas pelo cliente são recebidas pelo *Listener* e o mesmo passa a conexão para um processo despachante (*dispatcher*), que irá gerenciar as conexões entre os processos clientes e os processos servidores disponíveis. Nesse modo, a escalabilidade de usuário é garantida, podendo suportar um número maior de usuários conectados simultaneamente. Também oferece um melhor aproveitamento do uso da memória (ORACLE, 2005).

O despachante é um processo que manipula o gerenciamento das conexões para o processo servidor disponível. Pode suportar múltiplas conexões concorrentes de clientes (ORACLE, 2005).

O Processo Servidor (*Server process*) manipula a recuperação e armazenamento das informações no Banco de Dados, e meramente cuida do processamento de CPU necessário para uma aplicação (ORACLE, 2005).

3.3.3 *Oracle Net Configuration Assistant*

É uma ferramenta que oferece uma interface gráfica que permite ao usuário criar, alterar, remover e renomear os componentes básicos de rede do *Oracle Net*. Essa ferramenta oferece uma orientação passo a passo em qualquer configuração de componente que o usuário desejar. As configurações oferecidas são:

- Configuração do *Listener*.
- Configuração dos Métodos de Nomeação.
- Configuração do Nome de Serviço de Rede Local.
- Configuração do Uso de Diretórios.

3.3.4 *Oracle Net Manager*

O *Oracle Net Manager* é uma ferramenta de interface gráfica que permite as mesmas configurações que a *Oracle Net Configuration Assistent* permite. A única diferença é que essa ferramenta é a única que permite o usuário configurar um servidor *Oracle Names*.

O *Oracle Net Manager Wizard* é uma ferramenta *do Oracle Net Manager* que auxilia o usuário, passo-a-passo, na configuração das opções disponível pelo *Oracle Net Manager*.

Essa ferramenta é utilizada no processo de configuração do Banco de Dados Distribuído.

3.4 *Link* de Banco de Dados (*Database Links*)

O *Link* de Banco de Dados é o principal conceito em sistemas de Banco de Dados Distribuídos. Eles são responsáveis pela comunicação entre dois Bancos de Dados servidores,

permitindo que um cliente acesse um Banco de Dados servidor de forma transparente parecendo para o mesmo um Banco de Dados local.

Um *Link* de Banco de Dados é uma conexão de um único sentido, pois se um usuário do banco de dados "A" cria um *Link* de Banco de Dados que acessa um Banco de Dados "B", então o *Link* criado em "A" poderá acessar as informações contidas em "B", mas o inverso não poderá ocorrer utilizando o mesmo *Link* de Banco de Dados. Para que "B" acesse informações de "A", o usuário do Banco de Dados "B" deverá criar um *Link* de Banco de Dados em "B" para acessar "A".

Os *Links* de Banco de Dados são públicos ou privados. Um *Link* de Banco de Dados é chamado público quando na criação do mesmo a palavra PUBLIC é informada, tornando-o disponível para todos os usuários que se conectarem no sistema. Um l*ink* é privado quando a palavra PUBLIC é omitida, fazendo com que somente o usuário que o criou tenha acesso a ele.

Usuários do *link* de banco de dados podem acessar os bancos de dados remoto de três formas diferentes.

- *Link* do usuário conectado: usuários conectam ao Banco de Dados remoto, utilizando os mesmos dados de sua conta no Banco de Dados local. Nesse tipo de conexão o usuário deverá ter uma conta de acesso no Banco de Dados remoto.
- *Link* de usuário fixo: na criação do *Link* de Banco de Dados é informado o usuário e a senha que deverão ser utilizados para a comunicação com o Banco de Dados remoto. O usuário informado deve ter uma conta no Banco de Dados remoto.

- *Link* do usuário corrente: o usuário conecta como um usuário global e pode utilizar objetos criados por outros usuários. Usuários têm acesso a objetos de outros usuários e podem utilizar esses objetos no Banco de Dados remoto.

Um *Link* de Banco de Dados permite o acesso do usuário de um Banco de Dados local a um Banco de Dados remoto mesmo se o usuário do BD local não é um usuário do BD remoto.

Os nomes utilizados pelo *Link* de Banco de Dados deve ser o mesmo nome do Banco de Dados global ao qual o usuário deseja acessar ou pode ser qualquer nome caso o parâmetro de inicialização *GLOBAL_NAMES* seja alterado para *FALSE*. Por *default* Oracle inicializa o parâmetro como *TRUE* fazendo com que o nome do *link* seja o mesmo nome do Banco de Dados global. A sintaxe para a utilização de nomes globais é da seguinte forma: `<nome_do_banco_de_dados.dominio_da_rede>`. Considerando um usuário conectado em um Banco de Dados com nome "`A`" e nome global `A.BR.ORACLE.COM`, e considerando que ele deseja criar um *link* para o banco de dados "`B`" que está no domínio `US.ORACLE.COM`, então este usuário deverá criar o *link* de Banco de Dados com o seguinte nome: `B.US.ORACLE.COM`.

O nome de banco de dados global do Oracle segue uma hierarquia na qual o Banco de Dados está localizado nas folhas de uma árvore e, acima deles, está localizado o domínio da rede na qual identifica o *host* que contém o Banco de Dados. Considerando o Banco de Dados "`A`" e o Banco de Dados "`B`" citados na Figura 3.2 ilustra-se como são formados os nomes globais dos Bancos de Dados.

Na Figura 3.2 apresenta-se que o Banco de Dados "`A`" está localizado sob o domínio `BR`, que vem sob o domínio `ORACLE` que, por sua vez, está localizado no domínio `COM`. O mesmo ocorre com o Banco de Dados "`B`", exceto pelo fato de que o mesmo está localizado em outro domínio de rede.

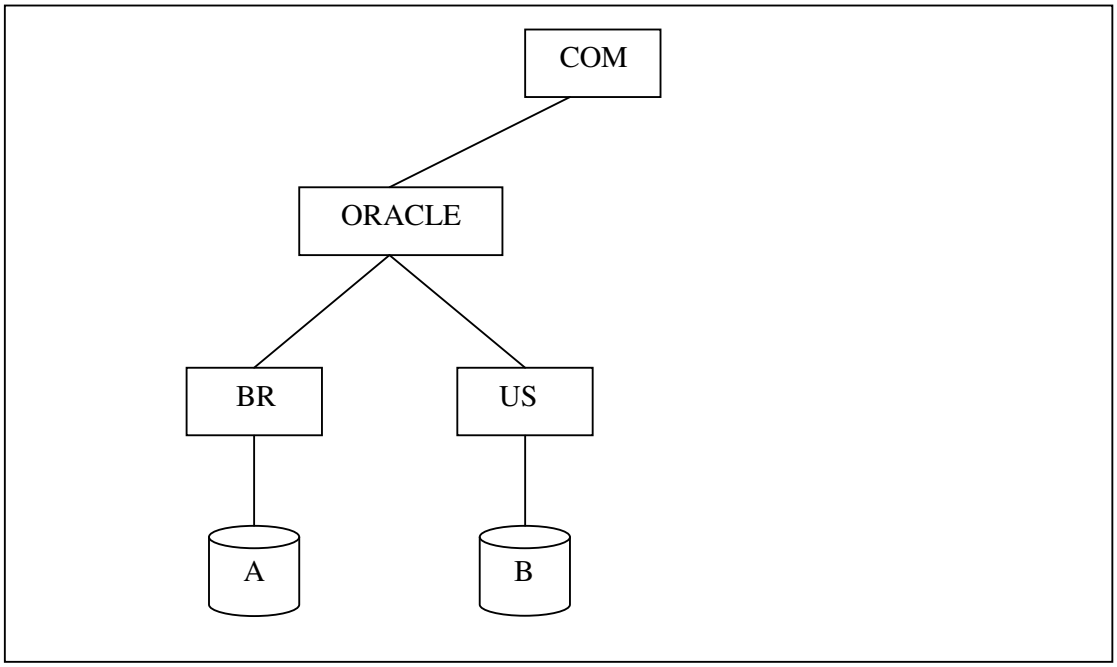


Figura 3.2 - Hierarquia de nomeação global Oracle (ORACLE, 2005).

Para um usuário do Banco de Dados "A" acessar um objeto do Banco de Dados "B", o usuário de "A" deverá ter um *Link* de banco de dados criado que referencia "B" e deverá utilizar a sintaxe <objeto_do_esquema@nome_do_banco_de_dados_global>, Onde objeto_do_esquema é uma tabela, índice, visão, sinônimo, procedimento, pacote, ou *Link* de Banco de Dados. Se "A" deseja recuperar informações da tabela remota EMP do Banco de Dados "B", então o mesmo deverá utilizar a seguinte instrução SQL:

```
SELECT * FROM EMP@B.US.ORACLE.COM,
```

onde B.US.ORACLE.COM é o nome do Banco de Dados remoto, ou seja, o nome do *Link* de Banco de Dados armazenado em "A".

Os nomes dos Bancos de Dados que compõem o sistema podem ser idênticos, mas o nome do Banco de Dados Global deve ser único para cada *host* da rede (ORACLE, 2005).

3.5 Sinônimos

Sinônimos são fundamentais para se obter transparência de localização. Um sinônimo permite ao usuário de um Banco de Dados acessar objetos de outro Banco de Dados com a mesma sintaxe que o usuário utilizaria para acessar informações no Banco de Dados ao qual está conectado (ORACLE, 2005).

A instrução SQL apresentada anteriormente mostra o acesso a uma tabela remota especificando o nome da tabela e o nome do *Link* de banco de dados que tem o mesmo nome do Banco de Dados remoto. Um usuário que dispõe de um sinônimo EMP configurado no Banco de Dados local e com permissão de acesso para o mesmo pode acessar dados no Banco de Dados remoto sem ter conhecimento da sua localização como mostra a seguinte sintaxe:

```
SELECT * FROM EMP.
```

Assim, sinônimos em conjunto com *Links* de Banco de Dados, criam a transparência de localização no SGBD Oracle.

4 - INSTALAÇÃO E CONFIGURAÇÃO DO SGBDD ORACLE

A partir deste capítulo são descritos os procedimentos realizados durante o projeto, que são apresentados na Figura 4.1.

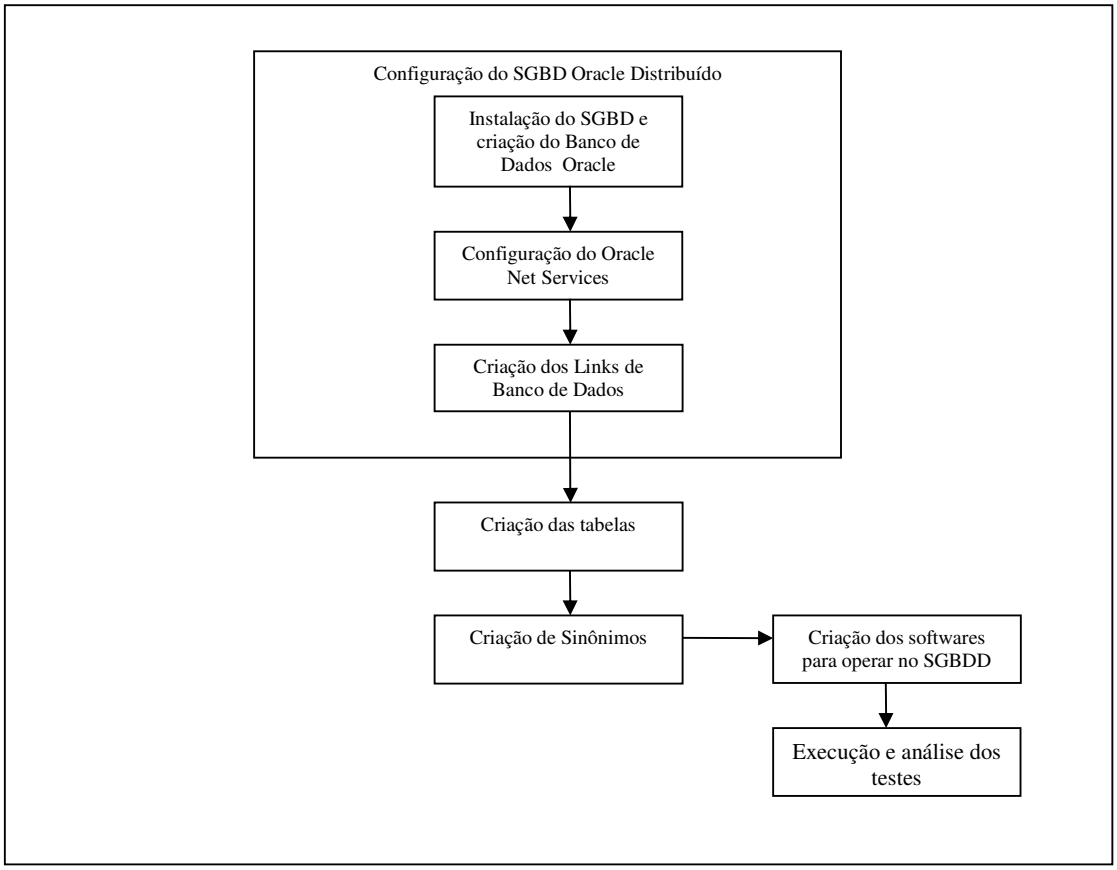


Figura 4.1 - Diagrama dos procedimentos realizados no projeto.

A primeira fase do trabalho, apresentada neste capítulo, consiste em configurar o SGBDD Oracle de acordo com os conceitos apresentados no capítulo anterior. A segunda parte do trabalho engloba o desenvolvimento de programas cujas funções são armazenar e recuperar imagens médicas e será descrita no Capítulo 5.

O processo de implementação do SGBD Oracle Distribuído será divididos em duas partes: a primeira parte descreve a instalação do Oracle e a criação do Banco de Dados. Na segunda parte serão mostradas as configurações necessárias para a distribuição dos dados.

4.1 Instalação do Oracle e criação do Banco de Dados

De acordo com Oracle (2005), a máquina deve concordar com as exigências mínimas para a instalação do SGBD Oracle9i. As exigências mínimas do SGBD Oracle9i são:

- Sistema Operacional: Windows 98/NT/2000/XP.
- Sistema de Arquivos: Oracle recomenda utilizar FAT 32 para Windows 98, e para os demais utilizar NTFS.
- Hardware: 128 MB de RAM(recomendado 256MB), adaptador de vídeo de 256 cores.
- Espaço em disco livre: Para o sistema de arquivo FAT é necessário 4.75GB de espaço disponível, já para a partição NTFS é necessário 2.85GB.
- Protocolos: TCP/IP, TCP/IP com SSL e Named Pipes.

Dando inicio à instalação, a tela mostrada na Figura 4.2 será exibida.

A Figura 4.3 exibe a tela de boas vindas do Oracle, através da qual é possível escolher entre eliminar produtos já instalados, visualizar produtos já instalados, abandonar a instalação e dar seqüência na instalação. A tecla `Próximo` deverá ser pressionada para dar seqüência ao processo.

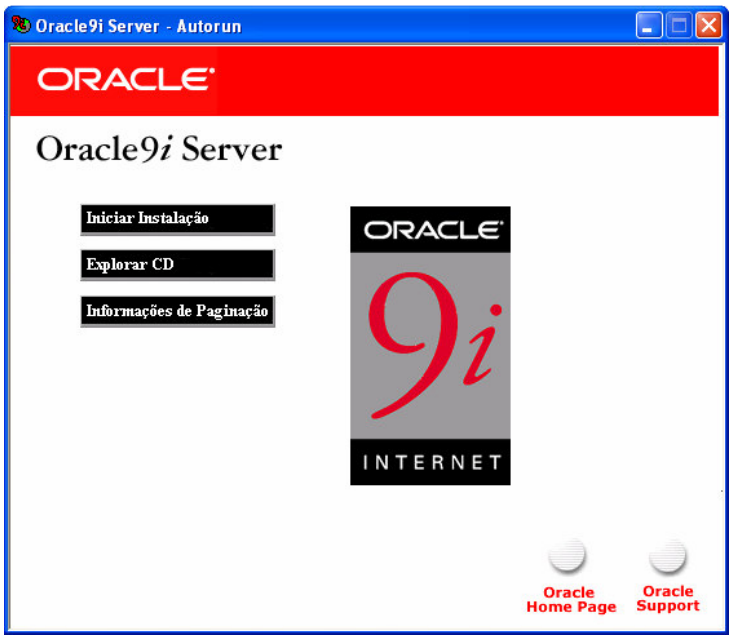


Figura 4.2 - Tela inicial da Instalação do Oracle.

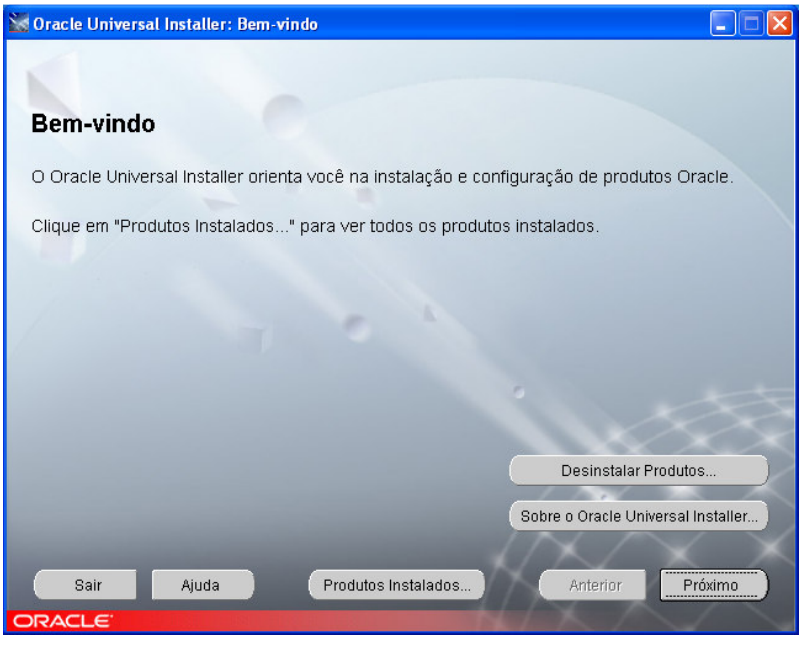


Figura 4.3 - Tela de boas vindas.

Ao dar seqüência no processo de instalação, a tela apresentada pela Figura 4.4 será exibida. Deverá ser informado o local onde estão os arquivos que contêm os produtos Oracle a serem instalados. No caso do presente trabalho, foi informada a localização dos produtos na máquina remota LAB0600.

O software de instalação procura no sistema local alguma instância existente do oracle. Caso nunca tenha sido instalado o Oracle na máquina, é sugerido um nome alternativo (OraHome92) para o Oracle Home, e um caminho padrão (C:\oracle\ora92) para a instalação dos produtos.

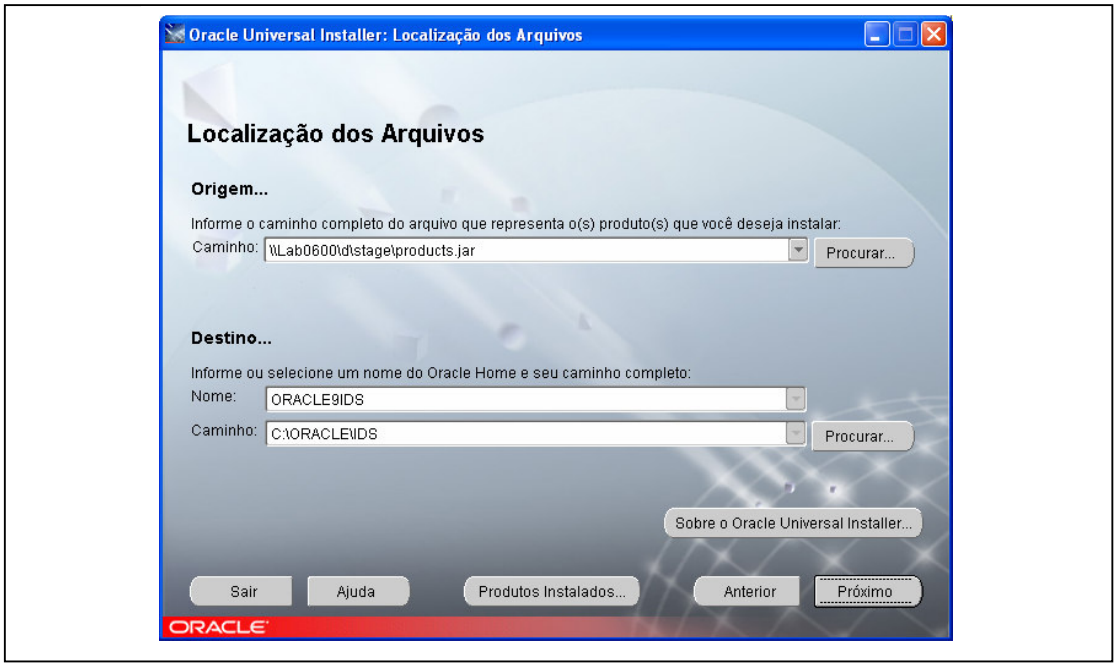


Figura 4.4 - Informações referente à instância do Oracle e ao local de instalação.

A Figura 4.5, mostra a tela na qual deve ser selecionado o tipo do produto que deseja ser instalado de uma lista de três opções.

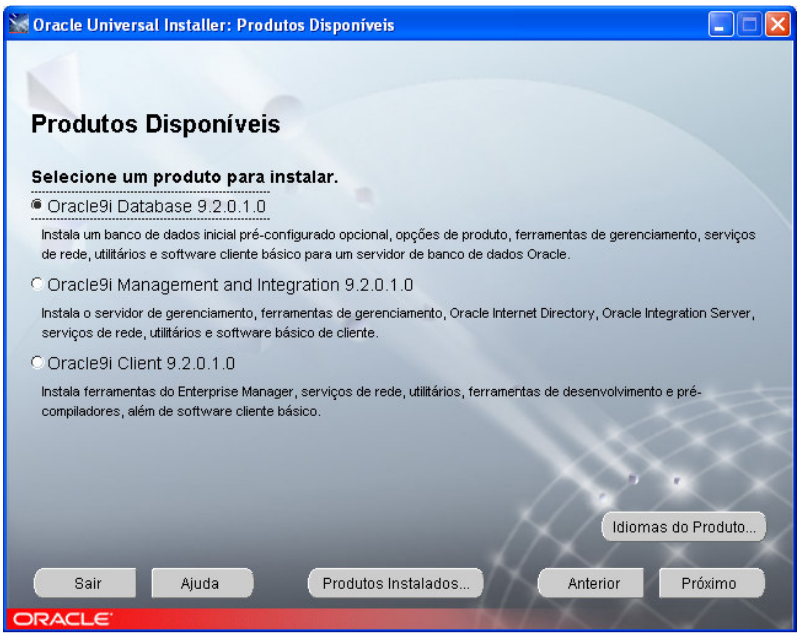


Figura 4.5 - Opções de produtos Oracle.

No presente trabalho a opção escolhida foi a primeira que disponibiliza um banco de dados pré-configurado, uma instância do Oracle e Serviços de Rede (*Oracle net Services*), que será utilizada para a distribuição dos dados.

A Figura 4.6 exibe os tipos de instalação disponíveis, o usuário pode escolher uma das três primeiras opções ou a opção *Personalizar* para selecionar os produtos que o mesmo deseja instalar.

A opção *Enterprise Edition* instala um banco de dados pré-configurado, serviços de rede, ferramentas para o ambiente de banco de dados, o *Oracle Enterprise Manager Framework* das ferramentas de gerenciamento, incluindo o Console, Gerente do servidor, e o *Intelligent Agent*, *Oracle Utilitis*, e a documentação online.

Na opção *Standard Edition* é instalado um banco de dados pré-configurado, serviços de rede, *Oracle Enterprise Manager framework* das ferramentas de gerenciamento, incluindo o Console, Gerente do servidor, e *Intelligent Agent*, e *Oracle Utilitis*.

Na terceira opção são disponibilizados os mesmos softwares da primeira opção, mas suportando apenas um único usuário que tenha compatibilidade total com as outras duas opções.

A quarta e última opção disponibiliza opções para criar um banco de dados que atenda às necessidades exclusivas do ambiente do usuário. Essa opção deve ser selecionada em caso de implementação de Sistema de Banco de Dados Distribuídos Heterogêneos.

A opção escolhida nessa etapa foi a primeira (*Enterprise Edition*) que atende a necessidade para a configuração do sistema proposto.

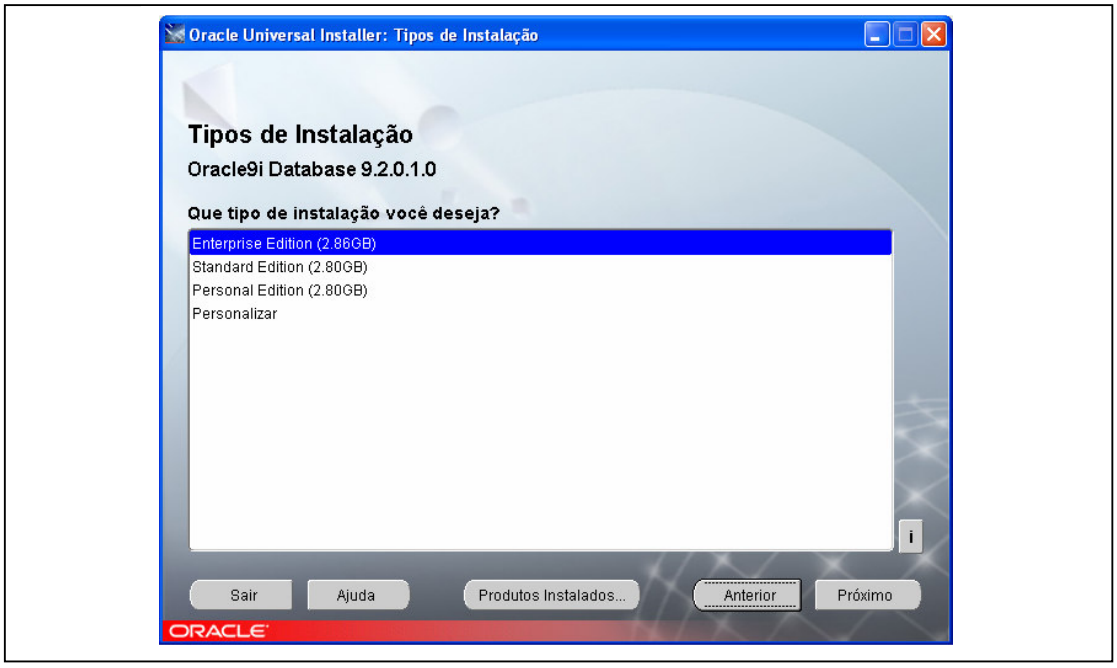


Figura 4.6 - Tipos de instalação oferecidos pela Oracle.

Nessa etapa da instalação, se alguma das opções de Banco de Dados oferecida pelo Oracle, como mostrado na Figura 4.7, atender as necessidades do usuário, então esta poderá ser selecionada. Caso o usuário queira instalar o Banco de Dados configurando os parâmetros necessários e escolhendo o que for necessário a ele, então deverá selecionar a opção

Personalizada. Para cada opção, exceto para *Personalizada* e *Somente Software*, o Oracle configura e otimiza um Banco de Dados para atender as necessidades do usuário:

- *Finalidade Geral*: o Banco de Dados criado nessa opção será otimizado para uso nas diversas ocasiões disponíveis.
- *Processamento de Transações*: configura o Banco de Dados de forma a oferecer uma otimização no processamento de transações.
- Data Warehouse: otimiza o Banco de Dados criado para operar como um *Data Warehouse*.
- *Personalizada*: o usuário é quem decide como deve ser instalado o Banco de Dados, e qual a melhor forma para a otimização do mesmo.
- *Somente Software*: instala somente os software, caso o usuário queira instalar um Banco de Dados mais tarde, deverá cria-lo utilizando o *Database Configuration Assistant*.

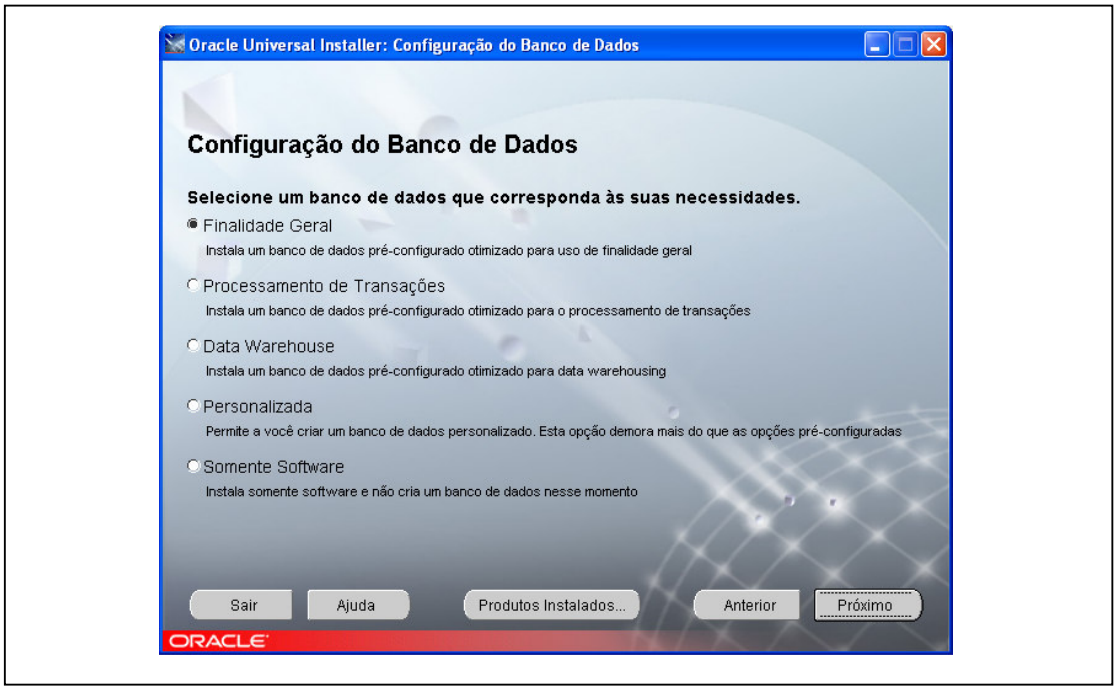


Figura 4.7 - Tipos de configurações disponíveis pelo Oracle.

As próximas duas etapas são de extrema importância. Nessa etapa deverá ser informado a porta à qual o *Listener* irá utilizar para esperar por conexões entrantes. No presente trabalho a porta 1521 foi a porta escolhida para o *Listener* atender por conexões do cliente. As três máquinas participantes do SGBDD foram configuradas para utilizar a porta 1521, mas não necessariamente elas devem ter o *Listener* configurado para ouvir o mesmo número da porta. Em caso de alteração do número da porta, na configuração do Nome *de Serviço da Rede Local* o número dessa porta deverá ser informado. A Figura 4.8 demonstra a escolha da porta padrão (*default*) 1521.

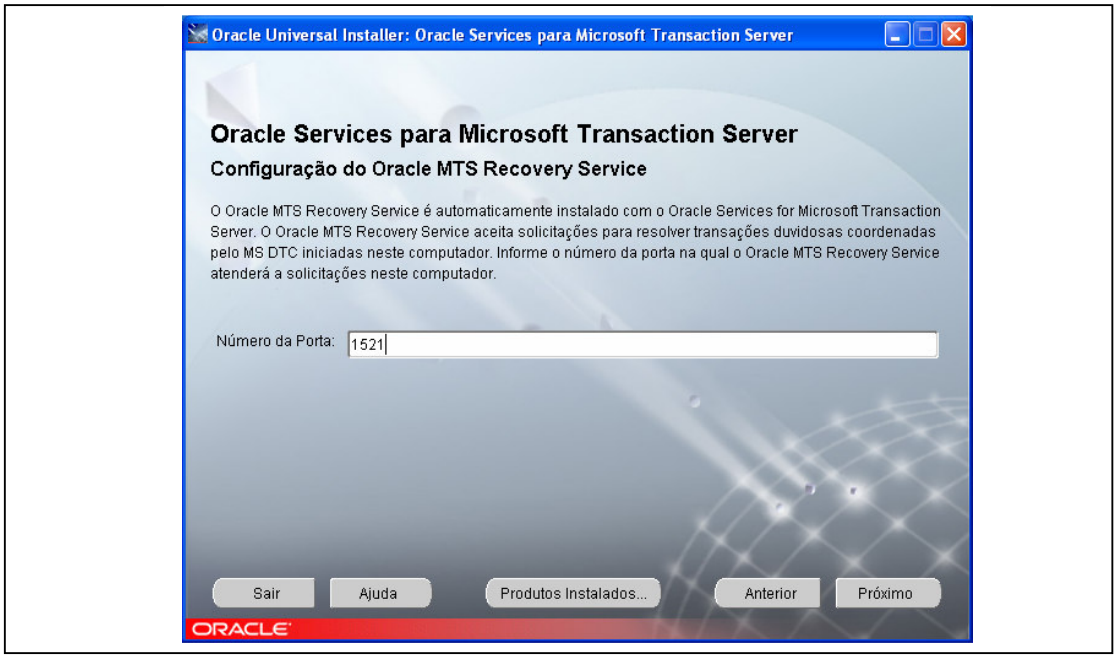


Figura 4.8 - Número da porta ao qual o *Listener* irá atender.

Nessa etapa deve ser informado, ainda, o nome do banco de dados global que é formado como a sintaxe mostrada na Seção 3.4, Capítulo 4, como também deve ser informado um SID. Um SID (*System Identifier Database*) é um identificador exclusivo do banco de dados, não pode ser informado um identificador que já esteja em uso pelo Oracle (ORACLE, 2005).

Ao colocar o nome do Banco de Dados global um SID será informado podendo ser aceito, ou informado outro. É recomendado que o usuário coloque seu SID, caso tenha outro Banco de Dados instalado na máquina, pois o Oracle não informa se o SID já está em uso. Se existe um Banco de Dados com o SID executando na máquina o Oracle emitirá um erro durante o processo de instalação.

O processo de informação do nome do Banco de Dados Global e do SID pode ser observada na Figura 4.9.

Figura 4.9 - Tela de Identificação do Banco de Dados.

O nome do banco de dados global informado no presente trabalho para a máquina LAB0623 é CARLOS.LAB0623.ORACLE.COM, onde CARLOS é o Nome do Banco de Dados e LAB0623.ORACLE.COM é o domínio da rede.

As configurações adotadas para cada *host* da rede foram:

1. Máquina LAB0623:

    - Nome do BD Global: CARLOS.LAB0623.ORACLE.COM.

- SID: CARLOS.

2. Máquina LAB0624:
    - Nome do BD Global: IMGMED.LAB0624.ORACLE.COM.
    - SID: IMGMED.
3. Máquina LAB0625:
    - Nome do BD Global: IMGMED.LAB0625.ORACLE.COM.
    - SID: IMGMED.

É importante salientar que em duas máquinas os bancos de dados possuem o mesmo nome, mas domínios de rede diferentes.

Na etapa seguinte um local para a instalação dos arquivos do Banco de Dados deve ser informado. O Oracle recomenda que os arquivos do Banco de Dados e o *software* do mesmo sejam instalados em discos secundários (*HDs*) diferentes. No presente trabalho foi informado o caminho `C:\oracle\oradata`, como mostra a Figura 4.10.

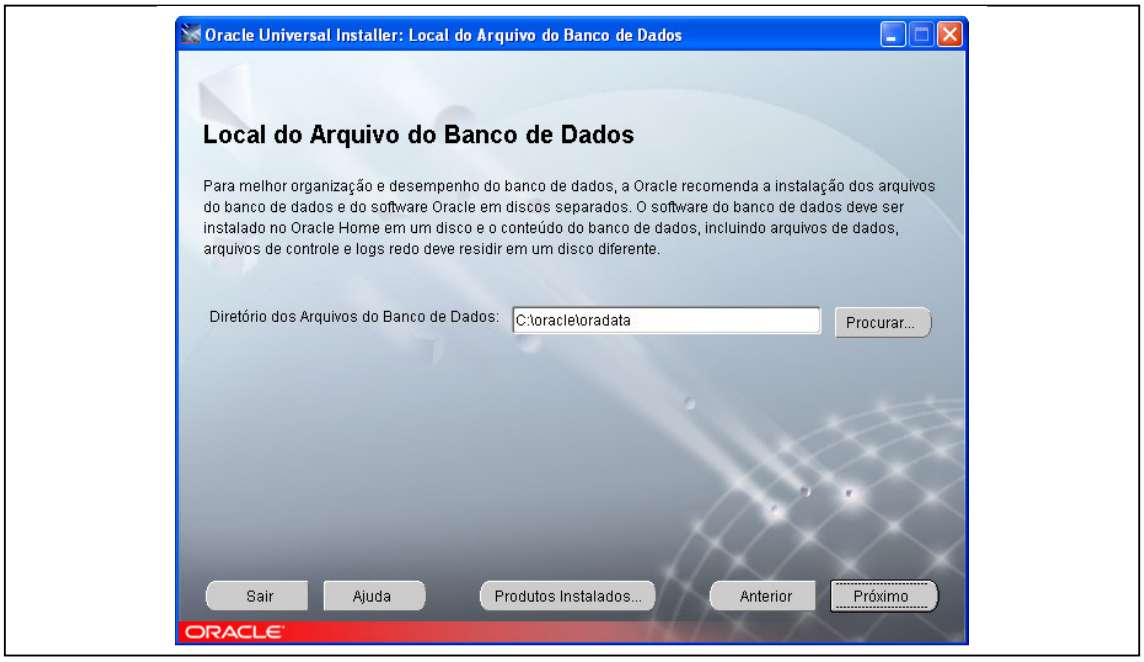


Figura 4.10 - Localização dos arquivos do Banco de Dados.

Na seqüência, é exibida uma tela para a escolha do conjunto de caracteres que o Banco de Dados utilizará. Como exibe a Figura 4.11, o conjunto de caracteres selecionado nessa etapa foi o conjunto *default* (primeira opção).

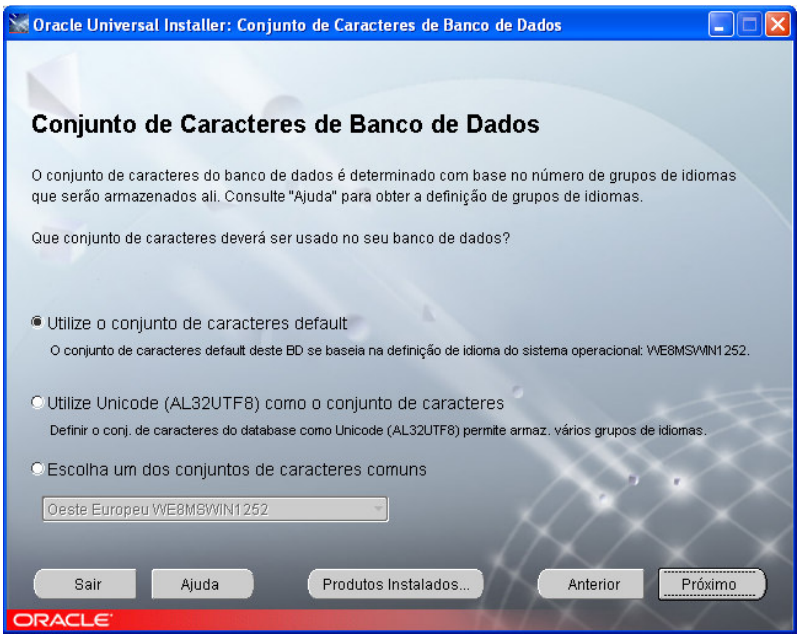


Figura 4.11 - Escolha do conjunto de caracteres de Banco de Dados.

No passo seguinte, o Oracle apresentará um resumo dos *softwares* e opções escolhidas a serem instalados na máquina do usuário, como mostrado na Figura 4.12. Dando seqüência ao processo inicia-se o procedimento de instalação das opções escolhidas pelo usuário.

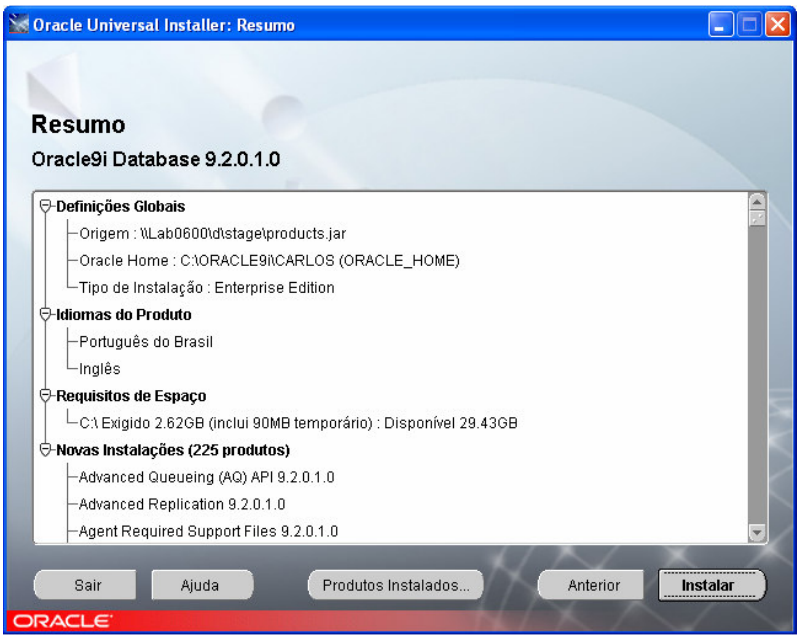


Figura 4.12 - Resumo das opções selecionadas.

Após o processo de instalação e durante o processo de configuração do Banco de Dados, a tela apresentada na figura 4.13 será exibida. Nesse passo da configuração do Banco de Dados deveram ser informadas as senhas para os usuários SYSTEM e SYS. Essa senha é de extrema importância, pois estes usuários têm permissão de acesso livre sem restrições ao Banco de Dados. O usuário SYSTEM será utilizado por toda a configuração do Banco de Dados Distribuído, e para o acesso ao editor SQL-PLUS[2] e a ferramenta *Enterprise Manager Console*[3].

[2] Editor utilizada para manipulação do banco de dados Oracle através de instruções SQL.
[3] Ferramenta gráfica do Oracle que permite o gerenciamento dos objetos do banco de dados.

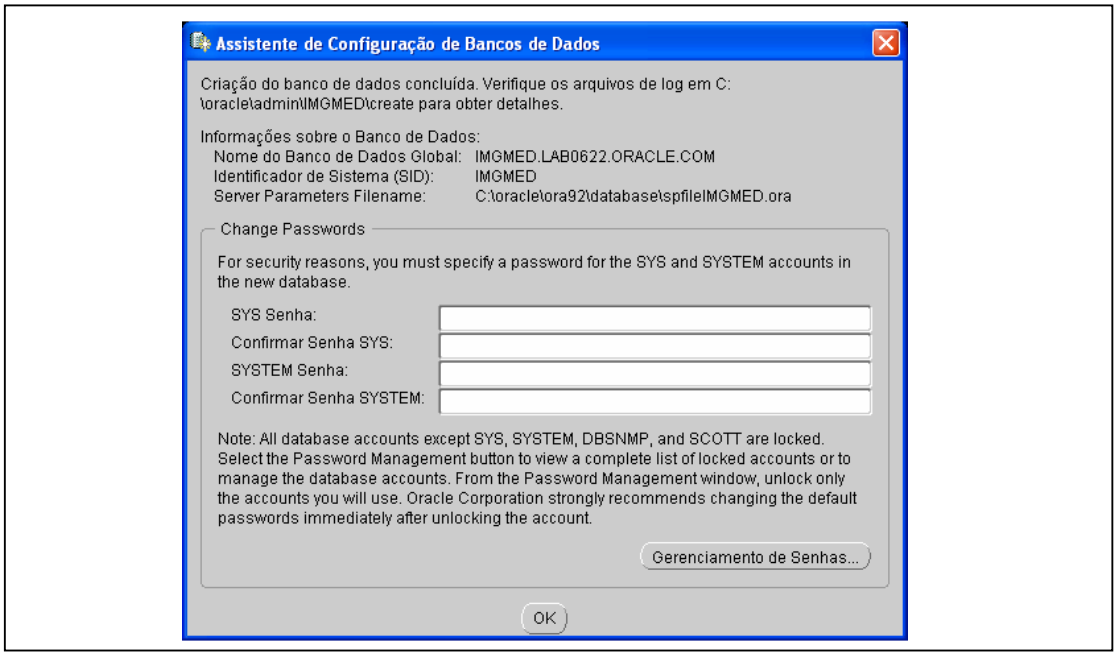


Figura 4.13 - Definição das senhas dos usuários: SYSTEM e SYS

Ao finalizar a instalação será mostrada a tela da figura 5.14. Nesse momento os softwares e o Banco de Dados já estão instalados e pré-configurados como o usuário optou nos passos anteriores.

Figura 4.14 - Finalização do processo de Instalação.

4.2 Configuração do Oracle Como SGBD Distribuído

Nessa etapa do projeto será configurado o Oracle para operar de forma distribuída. As ferramentas *Net Manager*, *Net Configuration Assistant* e *SQL-PLUS* são utilizadas para este processo.

Deve ser levado em consideração nesse momento que algumas das máquinas já possuíam uma instalação do *Oracle Client*, portanto foi refeita outra instalação sobre a existente. A instalação do Oracle nas máquinas LAB0624 e LAB0625 não foi como mostrado na seção anterior, fazendo com que maiores esforços fossem necessários na configuração. Já a máquina LAB0623 apresentou problemas depois de instalado o SGBD Oracle, e foi necessário refazer a instalação de acordo com o citado na seção anterior. Na reinstalação foi necessária a remoção de todos os softwares Oracle instalados na máquina.

Algumas tarefas adicionais tiveram que ser executadas nos *hosts* LAB0625 e LAB0624, tarefas como: criação de um novo *Listener*, alteração do nome do Banco de Dados Global, criação de um Serviço do Banco de Dados que o *Listener* atenderá.

Alguns problemas foram encontrados durante o processo de configuração do SGBDD Oracle, e estes serão descritos ao decorrer do capítulo.

4.2.1 Alteração do Nome do Banco de Dados Global

Antes de configurar o *Listener* foi necessária a alteração do Nome do Banco de Dados Global das máquinas LAB0625 e LAB0624. A alteração do Nome do Banco de Dados Global procedeu utilizando a ferramenta SQL-PLUS, executando a seguinte sintaxe da SQL:

```
ALTER DATABASE RENAME GLOBAL_NAME TO
<NOVO_NOME_DO_BANCO_DE_DADOS_GLOBAL>;
```

onde `NOVO_NOME_DO_BANCO_DE_DADOS_GLOBAL` foi informado `IMGMED.LAB0625.ORACLE.COM`, que substituiu o antigo nome do Banco de Dados Global `IMGMED.ORACLE.COM` do *host* LAB0625. E para o *host* LAB0624 foi informado `IMGMED.LAB0624.ORACLE.COM` substituindo o antigo `IMGMED.us.oracle.com`.

O próximo passo foi a criação do *Listener* e configuração do Serviço de Banco de Dados utilizando a ferramenta *Oracle Net Manager*.

4.2.2 Configurando o Listener

O Oracle cria um *Listerner* que por padrão é chamado *LISTENER* após a instalação, atendo por conexões na porta 1521. Devido os *hosts* Lab0625 e Lab0624 terem sido configurados antes do *host* LAB0623, foi atribuído um novo *Listener* para cada *host* devido à existência de um Banco de Dados já instalado nas máquinas. Os novos *Listener* foi configurado para atender solicitações na porta 1521, e os números da porta dos Listeners existentes foram alterados para atender conexões na porta 1080.

Não é necessária a criação de um novo *Listener* para atender a um novo Banco de Dados, é necessária apenas a criação de um novo Serviço de Banco de Dados para o *Listener* corrente atender. Essa opção foi feita devido alunos utilizarem o laboratório durante as aulas.

Para criação de um *Listener* adicional, os seguintes passos foram executados:

1. Iniciar O *Oracle Net Manager*.
2. Selecionar o ícone *Listener*.
3. Fornecer o nome do *Listener*.
4. Selecionar o local de atendimento do *Listener*.
5. Clique em Adicionar Endereço.

6. Fornecer o nome do *host*, o protocolo, e a porta se necessário. Para a máquina LAB0625, o nome do *host* informado foi LAB0625, o número da porta escolhida para atender por conexões entrantes foi a 1521, e o protocolo TCP/IP, como mostra a figura 4.15.
7. Selecionar *Salvar Configurações* da rede no menu *Arquivo* do *Oracle Net Manager*.

Ao optar por utilizar o *Listener default* e apenas inserir um novo Serviço de Banco de Dados, o usuário deverá selecionar o *Listener* de nome *LISTENER*, selecionar a opção Serviços de Banco de Dados, Adicionar Endereço, e logo em seguida informar os dados do Banco de Dados no qual o *Listener* atenderá.

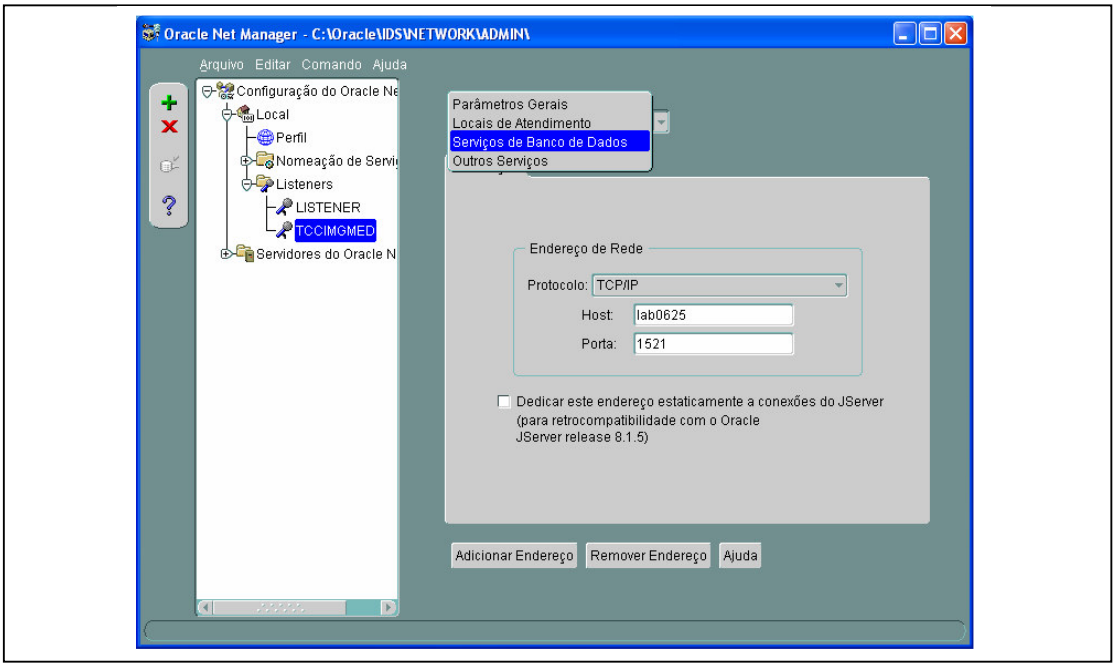


Figura 4.15 - Configuração do *Listener*.

4.2.3 Configurando os Serviços do Banco de Dados

Nessa configuração o usuário deve informar os dados necessários da instância do Banco de Dados local. O *host* LAB0624 é o *host* selecionado para apresentar a configuração dos Serviços do Banco de Dados e a criação dos *Links* de Banco de Dados e sinônimos. Para isso o usuário deverá seguir os seguintes passos.

1. Selecionar *Serviços de Banco de Dados* como é mostrado na figura 4.15.
2. Clicar em *Adicionar Banco de Dados*.
3. Entre com o Nome do Banco de Dados Global, o diretório do *Oracle Home*, e o *SID*, se necessário, nos campos apropriados.
4. Selecionar Salvar configurações de rede no menu arquivo do *Oracle Net Manager*.

A Figura 4.16 mostra o resultado da configuração do Serviço de Banco de Dados do *host* LAB0625.

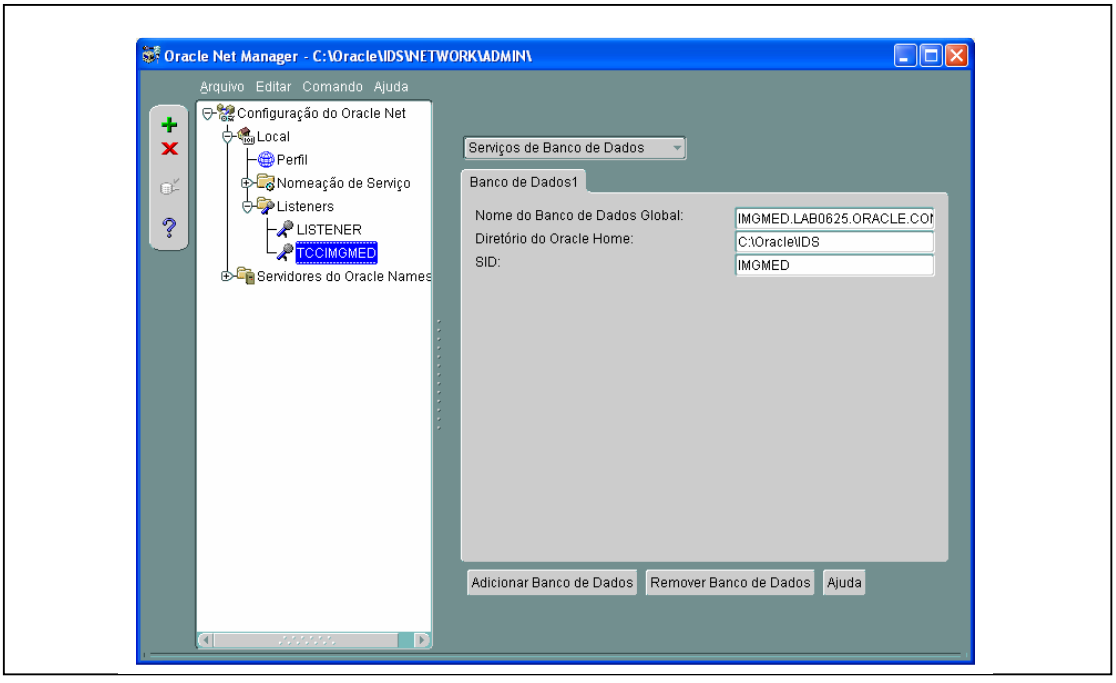


Figura 4.16 - Configuração final do Serviço de Banco de Dados.

. A máquina LAB0623 já estava com as configurações adequadas nesse passo. Para as outras máquinas houve a necessidade de ser informada uma configuração específica.

Como alternativa, um *Listener* pode ser configurado utilizando o *Oracle Net Configuration Assistant*. Para isso o usuário deverá executar os seguintes passos:

1. Iniciar o *Oracle Net Configuration Assistant*
2. Selecionar *Configuração do Listener* e clicar em `próximo`, como mostra a Figura 4.17.
3. Escolher a opção *Adicionar* (Figura 4.18). Caso nenhum *Listener* esteja configurado, as demais opções não estarão disponíveis.
4. Deverá ser informado o nome do *Listener* como mostra a Figura 4.19. Caso o nome exista, uma mensagem será exibida.

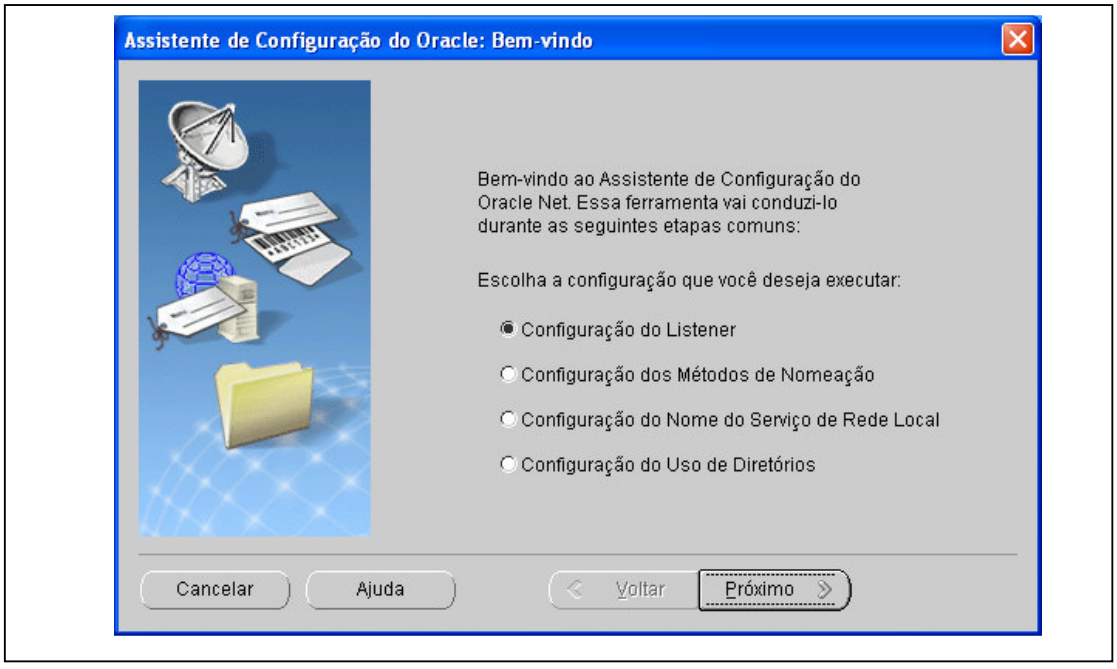


Figura 4.17 - Tela principal do *Oracle Net Configuration Assistant*.

Figura 4.18 - Adicionando um *Listener* com *Oracle Net Configuration Assistant*.

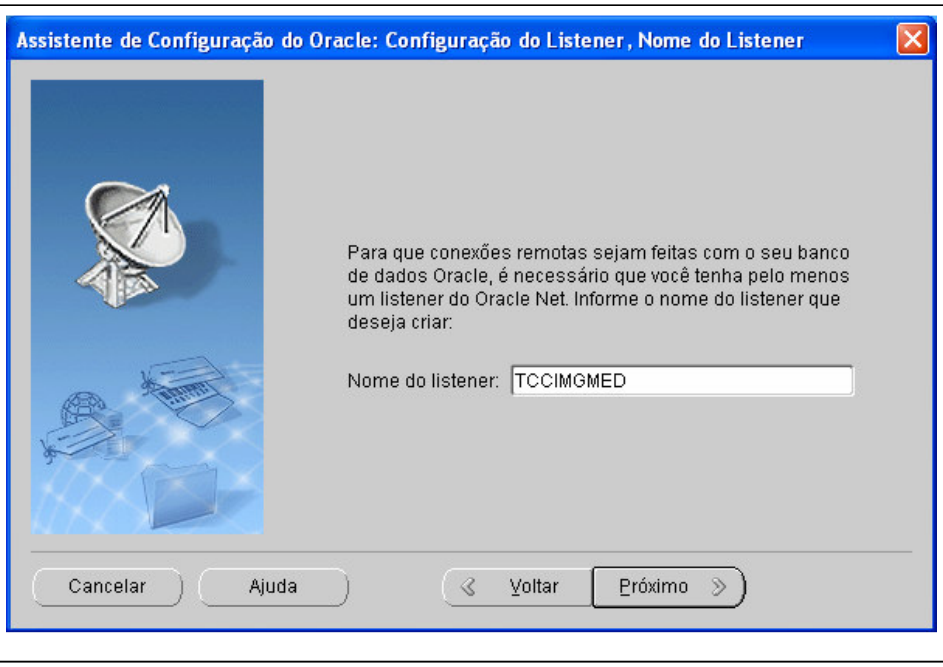


Figura 4.19 - Nomeação do *Listener*.

5. Selecionar o protocolo de comunicação utilizado (Figura 4.20).

Figura 4.20 - Selecionar protocolos de comunicação.

6. Selecionar a porta *default* (1521), ou informar um número de porta ao qual o *Listener* vai esperar por conexões entrantes (figura 4.21).

Figura 4.21 - Selecionar a porta de comunicação do *Listener*.

Ao final do processo, uma tela poderá aparecer perguntando qual *Listener* o usuário deseja inicializar, como é mostrado na Figura 4.22.

O Oracle disponibiliza ainda um comando para controlar o *Listener* a partir do *prompt* de comando fornecido pelo sistema operacional. A sintaxe do comando é a seguinte:

```
C:> LSNRCTL <Opção> [Nome do Listener]
```

Para o presente trabalho as opções utilizadas serão apenas *START* e *STOP* que inicia e finaliza um *Listener* respectivamente. Para iniciar um *Listener* o comando LSNRCTL START [Nome do *Listener*] deve ser executado.

A Figura 4.23 demonstra a utilização do comando de inicialização do *Listener* TCCIMGMED da máquina LAB0624 no *prompt* de comando do Windows XP.

Figura 4.22 - Escolha do *Listener* a ser inicializado.

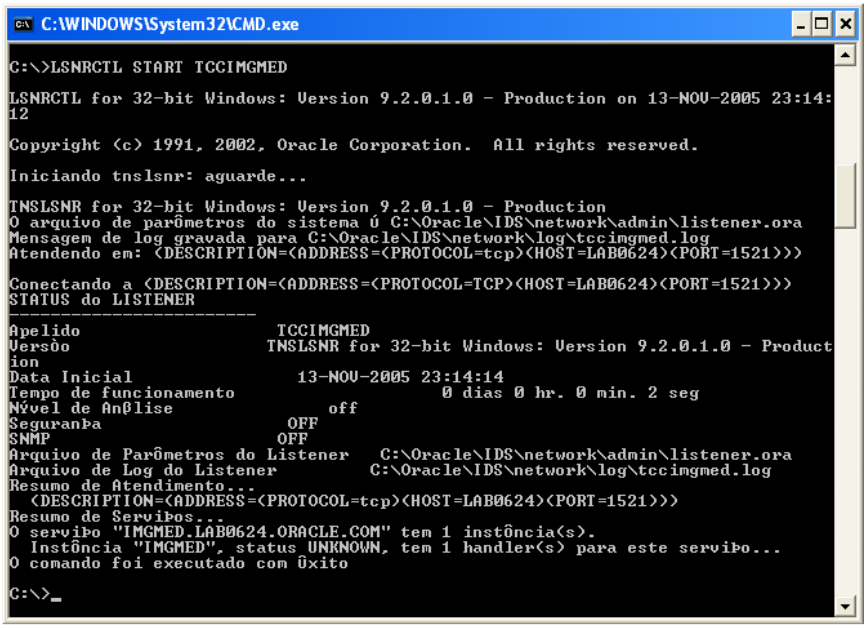


Figura 4.23 - Inicialização do *Listener* TCCIMGMED.

4.2.4 Configuração do Oracle Net Services

Nessa etapa são configurados os métodos de nomeação e os nomes de Serviços de Rede local.

4.2.4.1 Configurando os Métodos de Nomeação

Os métodos de nomeação disponibilizam cinco opções:

- *Nome do Host*
- *Local*
- *Oracle Names*
- *Sun NIS*
- *DCE CDS*

O Nome do *Host* elimina a necessidade de criar e manter um arquivo de configuração de nomes locais (*tnsnames.ora*), mas necessita de um mecanismo de tradução de endereços IP. Elimina a necessidade de entender processos de administração do *Oracle Names* ou *Oracle Internet Directory*. A conexão é feita utilizando apenas o nome do *host*. Esse método exige a instalação do protocolo TCP/IP tanto no cliente como no servidor, e também a instalação do *Oracle Net Services* no servidor (ORACLE, 2005).

O método de nomeação *Local* necessita que os nomes de serviços sejam armazenados no arquivo *tnsnames.ora* para que ele possa resolver os nomes de forma local. Na utilização desse método o Oracle recomenda que os nomes de serviços sejam atribuídos ao arquivo *tnsnames.ora* utilizando uma ferramenta gráfica como *Oracle Net Configuration Assistant*.

A Figura 4.24 demonstra a configuração dos métodos de nomeação.

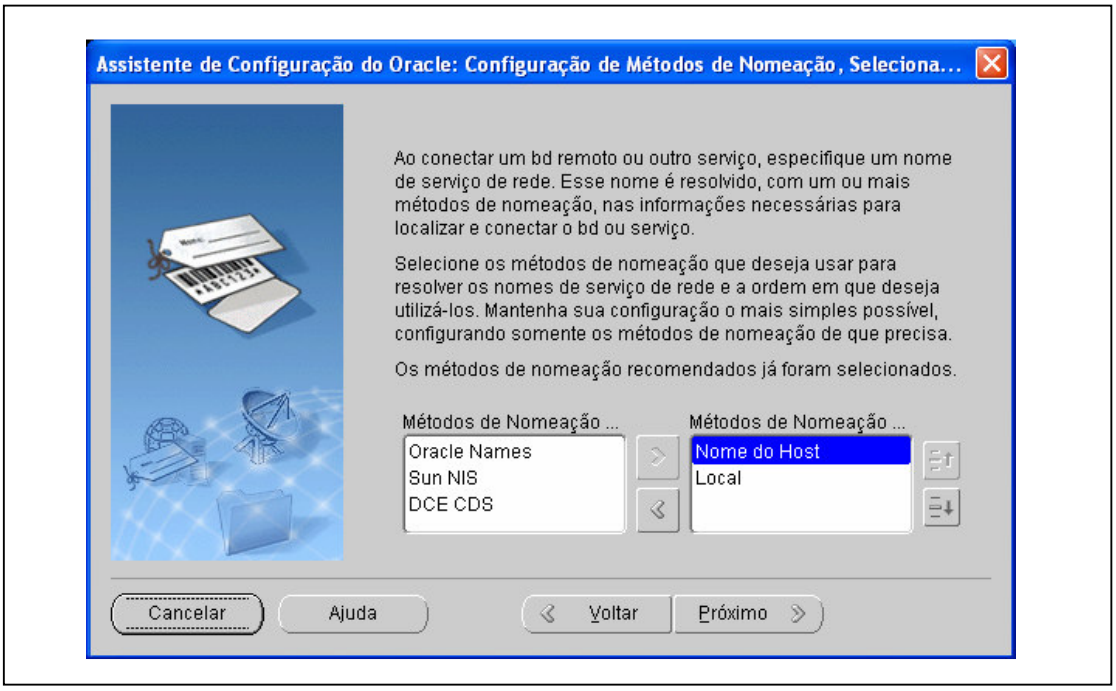


Figura 4.24 - Selecionando Métodos de Nomeação.

Por *default*, o Oracle seleciona *Nome do Host*, *Local*, e *Oracle Names*. Os métodos de nomeação que foram utilizados no presente trabalho são: *Nome Host* e *Local*. Após selecionar o método de nomeação local, a configuração do nome de serviço de rede local deve ser executada.

4.2.4.2 Configurando o Nome de Serviço de Rede Local

Nessa etapa foram configurados três nomes de Serviços de Rede Local, um para cada máquina pertencente ao SBDD. Nesse caso se a instalação do Oracle foi executada como mostra a Seção 4.1, Capítulo 4, não será necessário reconfigurar o *Nome de Serviço de Rede Local* no *host* local. É necessário apenas configurar o serviço para as máquinas remotas.

A descrição passo a passo da configuração do nome de serviço de rede local é apresentada a seguir:

1. Selecionar *Configuração do Nome de Serviço de Rede Local* (Figura 4.25).

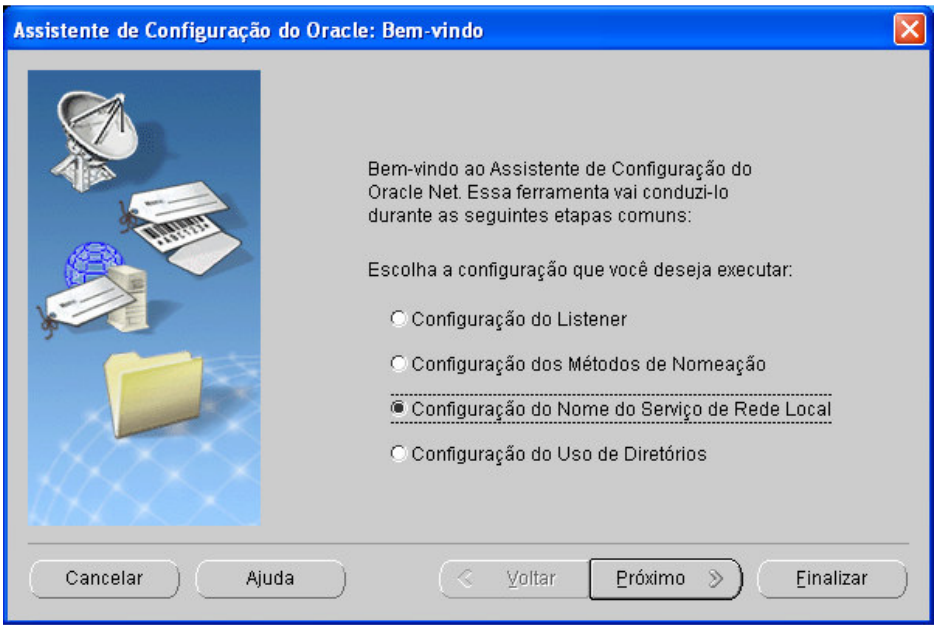


Figura 4.25 - Escolhendo Configuração do Nome do Serviço de Rede Local.

2. Selecionar adicionar, como mostra a figura 4.26.

Figura 4.26 - Escolhendo adicionar novo Nome de Serviço de Rede.

3. Escolher a versão do Banco de Dados Oracle que deseja acessar (Figura 4.27). Versões mais recentes do Banco de Dados Oracle necessitam de configurações extras no *Listener* (ORACLE, 2005).
4. Especificar o nome do serviço. Nesse momento deve-se informar o nome do Banco de Dados Global remoto ao qual deseja acessar. A Figura 4.28 demonstra a configuração de um nome de serviço para Banco de Dados `IMGMED.LAB0625.ORACLE.COM`, a partir do *host* LAB0624.
5. Selecionar um protocolo a ser utilizado para a comunicação entre os Bancos de Dados utilizados na rede (Figura 4.29).

Figura 4.27 - Escolha da versão do Banco de Dados.

Figura 4.28 - Informando o Nome do Serviço.

Figura 4.29 - Escolha do protocolo de comunicação.

6. Informar o nome do *host* e o *número da porta* de comunicação a ser utilizada. O campo nome do *host* deve ser preenchido com o nome do *host* remoto ao qual o Banco de Dados local irá conectar. Nesse caso o *host* remoto é a máquina LAB0625. O número da porta deverá ser o mesmo número da porta que o *Listener* foi configurado na máquina remota. Na instalação, a opção de porta foi a *default*, como mencionado na Seção 4.1, Capítulo 4. Nessa etapa deverá ser informada a mesma porta, no caso 1521 como mostra a Figura 4.30.

Figura 4.30 - Informando o Nome do *Host* e a porta de comunicação.

7. Executar o teste para verificar se há comunicação entre os *hosts*. Antes de executar o teste de conexão, o usuário deve assegurar que as informações fornecidas nesse processo de configuração estão configuradas no *host* remoto, e o processo *Listener* no *host* remoto esteja iniciado para estabelecer a conexão. Para garantir que o *Listener* esteja iniciado, deve se executar o comando para manipulação do *Listener* apresentado na Seção 4.2.3, Capítulo 4, (Figura 4.31).

Figura 4.31 - Teste de conexão.

Na Figura 4.32 é mostrado quando o teste de conexão foi bem sucedido.

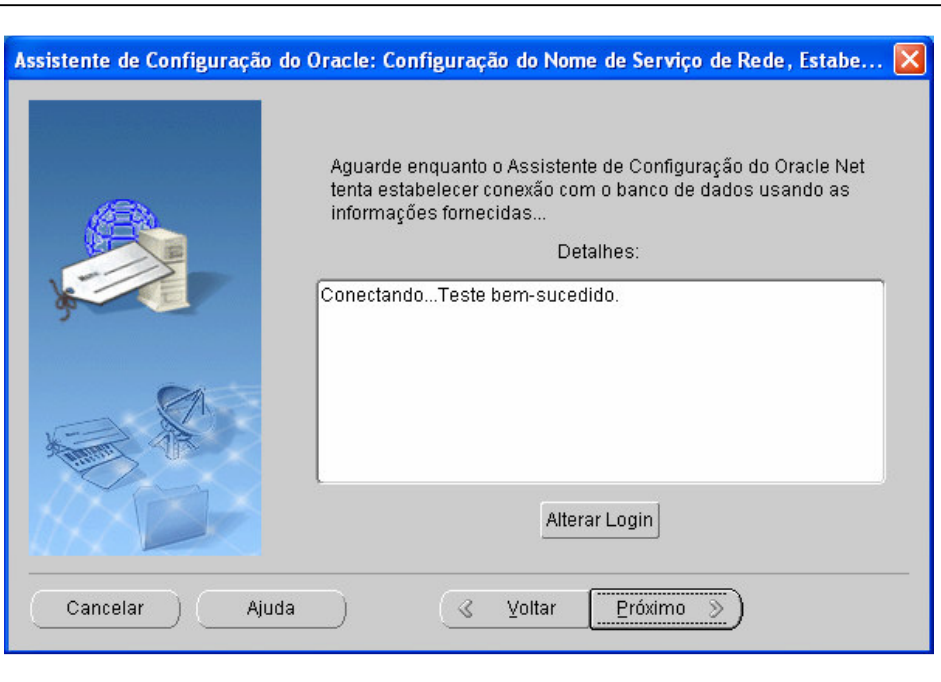


Figura 4.32 - Teste de conexão bem sucedida.

8. Fornecer um nome para o serviço configurado. No caso do presente trabalho, para cada nome de serviço configurado, foi adotado o esquema `nome_do_banco.nome_do_host` como mostra a Figura 4.32. Esse passo exige muita atenção, pois se um nome existente de serviço for substituído, o Banco de Dados pode não operar da maneira adequada.

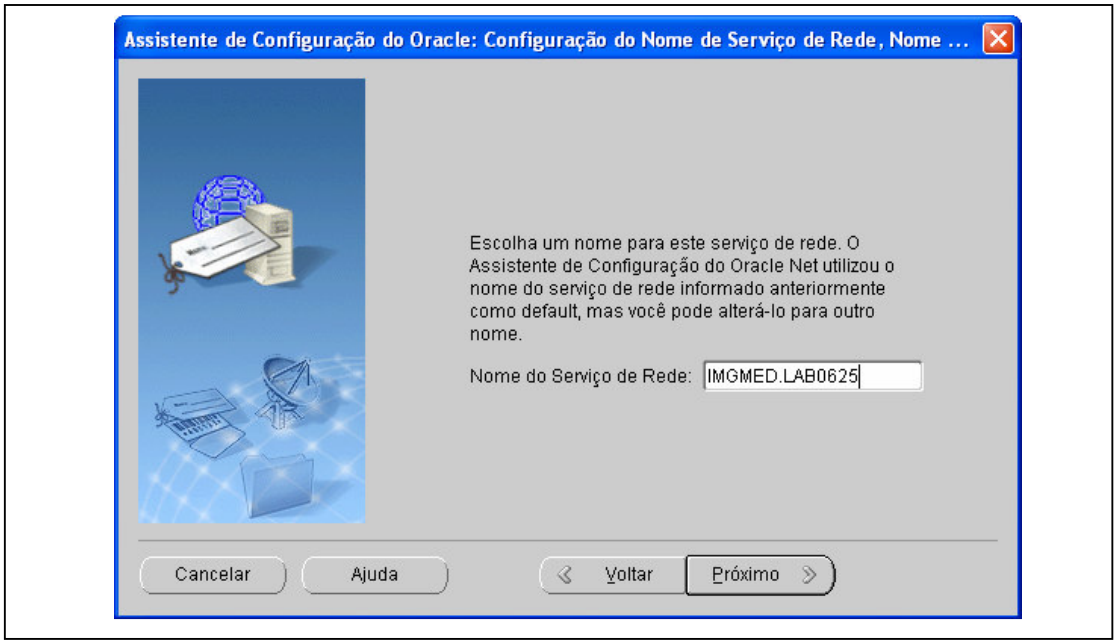


Figura 4.33 - Dando um nome ao serviço configurado.

Ao término da configuração, a ferramenta irá perguntar para o usuário se o mesmo deseja configurar outro nome de serviço.

Durante o passo 7 alguns erros surgiram, como *login* e senha do usuário inadequados e inexistência do *Listener*. O primeiro erro ocorre devido o *login* ou senha do usuário informado não conferir com a conta do mesmo no *host* remoto. Esse erro pode ser resolvido alterando-se esses atributos de forma adequada. Já o segundo ocorre se há falta de inicialização do *Listener* no *host* remoto ou local. Este erro pode ocorrer quando a Configuração do Serviço de Rede Local não está correta.

As telas dos erros podem ser encontradas no ANEXO K.

4.2.5 Configurando os Links de Banco de Dados

Para finalizar a configuração o usuário deve acessar a ferramenta SQL PLUS do Oracle, informando o seu *login* e senha. No presente trabalho o usuário utilizado foi o `SYSTEM`, que não tem nenhuma restrição de privilégios no Banco de Dados, economizando esforços de configuração de privilégios.

Após estabelecer a conexão deve ser executada a sintaxe da SQL mostrada na Figura 4.34 para criar os *Links* de Banco de Dados.

```
SQL> CREATE  DATABASE LINK "IMGMED.LAB0625.ORACLE.COM"
CONNECT
TO
  2      "SYSTEM"
  3      IDENTIFIED BY "MANAGER1"
  4      USING 'IMGMED.LAB0625' ;

SQL> CREATE  DATABASE LINK "CARLOS.LAB0623.ORACLE.COM"
CONNECT
  2  TO
  3       "SYSTEM"
  4       IDENTIFIED BY "MANAGER"
  5       USING 'CARLOS.LAB0623'
  6  ;
```

Figura 4.34 - Criando os *Links* de Banco de Dados.

A primeira sintaxe (Figura 4.34) apresenta a criação do *Link* de Banco de Dados para acessar o Banco de Dados remoto `IMGMED.LAB0625.ORACLE.COM`. Na sintaxe o usuário remoto fornecido é o usuário `SYSTEM` com a senha `MANAGER1`. A cláusula `USING` informa o nome de serviço local a ser utilizado, que foi configurado anteriormente. A segunda sintaxe

cria o *Link* de Banco de Dados para acesso ao *host* `CARLOS.LAB0623.ORACLE.COM` usando o serviço local `CARLOS.LAB623`.

Esse processo pode ser feito utilizando a ferramenta *Enterprise Manager Console* do Oracle. Para que o usuário faça essa configuração utilizando essa ferramenta, deverá executar os seguintes passos:

1. Escolher *Acionar independente* na tela de abertura da ferramenta.
2. Selecionar o Banco de Dados que o usuário configurou.
3. Informar o *login* e senha do ususário.
4. Selecionar Distribuição.
5. Selecionar Vínculos de Banco de Dados.
6. Selecionar *criar*, e a tela apresentada pela Figura 4.35 será exibida.

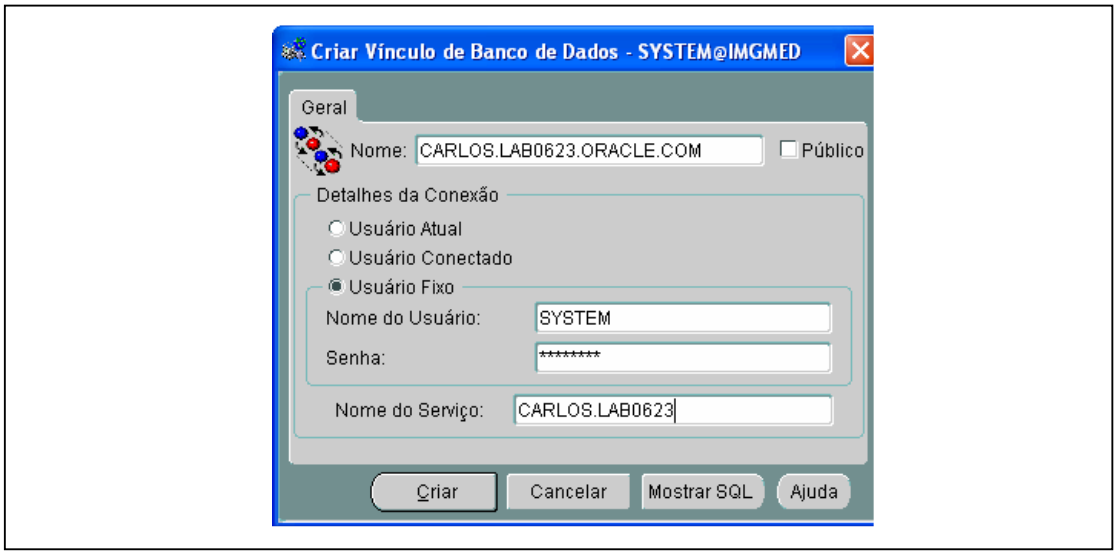


Figura 4.35 - Criando um *Link* de Banco de Dados para acessar o *host* remoto LAB0623.

7. O nome do Link de Banco de Dados deverá ser informado, devendo ter o mesmo nome do Banco de Dados remoto que ele referenciar. O tipo de usuário utilizado foi o usuário fixo descrito na Seção 3.4, Capítulo 3, o link

criado é um *link* privado, na qual somente o usuário informado no campo nome de usuário terá acesso a ele. O nome de serviço deve ser um nome de serviço já configurado para acesso ao *host* remoto.

8. Pressionar criar.

Se a configuração feita até este ponto estiver correta, nenhum erro será emitido. Caso contrário, se o serviço fornecido para um *link* remoto for o serviço do Banco de Dados local, será emitido um erro dizendo que um *link* de *loopback* deve ter um descritor de conexão associada a ele.

O processo deverá ser refeito para criar o *Link* de banco de dados para os outros *hosts* que participaram do sistema. Ao término da configuração dos *Links* de banco de dados, os mesmos poderão ser vistos na tela da ferramenta *Enterprise Manager Console* como mostra a Figura 4.36.

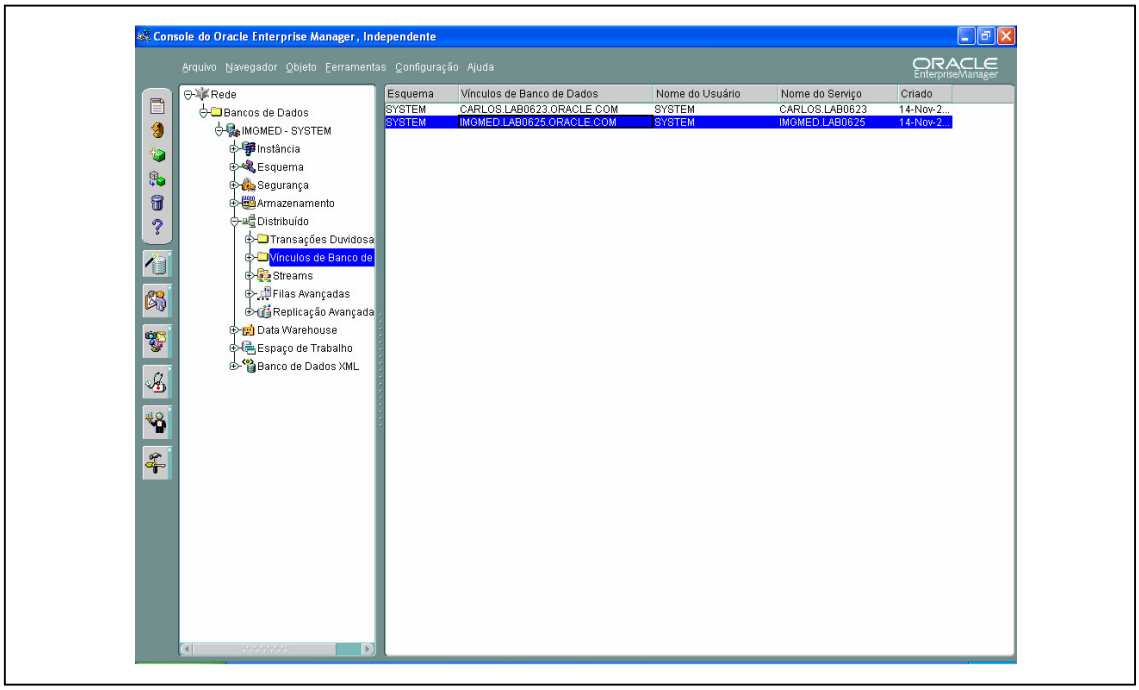


Figura 4.36 - ***Links*** **de Banco de Dados criados mostrados pelo** ***Enterprise Manager Console*****.**

Neste ponto o BD está pronto para ser usado de forma distribuída. No próximo capítulo será apresentada a metodologia utilizada para verificar a performance do Oracle para armazenar e recuperar dados de forma distribuída.

5 - METODOLOGIA DO SOFTWARE

Neste capítulo serão descritos os procedimentos adotados para a implementação do software que tem por objetivo armazenar e recuperar imagens médicas de tamanho variável utilizando dois tipos de dados: `BLOB`, que possibilita o armazenamento de dados em sua forma binária no banco de dados e `VARCHAR2,` que possibilita o armazenamento de um conjunto de caracteres no banco de dados. O armazenamento e recuperação serão executados de forma distribuída podendo acessar os dados no banco de dados local, como também nos bancos de dados remotos.

5.1 Apresentação do estudo de caso

Após a criação das tabelas e sinônimos no Banco de Dados, procedeu a geração do *software* para o armazenamento e recuperação das imagens médicas utilizando a linguagem de programação Java (SUN, 2005), desenvolvida pela *Sun Microsystem* em meados dos anos 90, baseado na Java 2 *Platform Standard Edition* 5.0 (JDK5.0), em conjunto com as classes fornecidas pelo SGBD Oracle. A API (*Application Programming Interface*) das classes Java fornecidas pelo Oracle pode ser obtida do arquivo *javadoc.zip*, localizada no diretório `\javavm\doc`, apartir do diretório do *Oracle Home* informado na Seção 4.1, Capítulo 4. O arquivo *classes111.zip*, localizado no diretório `jdbc\lib` a partir do diretório do *Oracle Home*, contém os arquivos *.class* que devem ser descompactados dentro do diretório `<raiz>:\<diretório_do_JDK5.0>\jre\Classes\`. Se o diretório `Classes` não existir o projetista deverá criá-lo.

Na Figura 5.1 é apresentado o diagrama de classe dos softwares desenvolvidos. As classes `InserirBlob`, `BuscarBlob`, `InserirPath` e `BuscarPath` são as classes que

iniciam os processos de inserção e recuperação instanciando um objeto da classe `OperacoesTCC`.

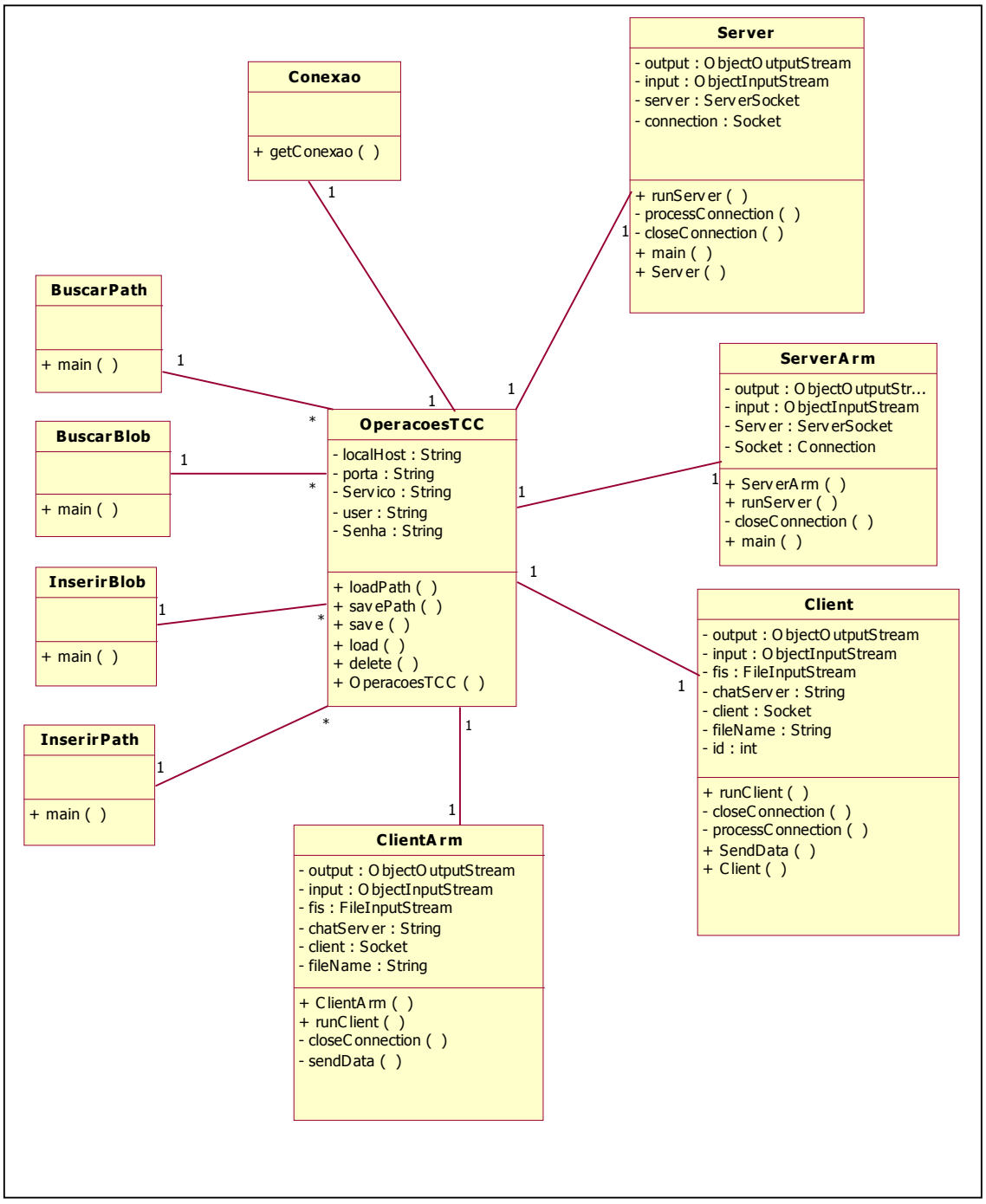


Figura 5.1 - Diagrama de classe do software desenvolvido

5.2 Criação de tabelas e sinônimos

Após a configuração do Banco de Dados Distribuído e da criação dos *Links* de Banco de Dados, é necessária a criação da tabela mostrada na Figura 5.1. Esse *script* mostra a tabela necessária para a realização dos testes. A tabela `IMAGEM` apresenta um campo do tipo `BLOB`, um campo do tipo `VARCHAR2` de 200 posições e um campo inteiro, que é a chave primária da tabela.

```
CREATE TABLE IMAGEM(
       ID INTEGER,
       URL VARCHAR2(200),
       BLOB_IMAGEM BLOB
);

ALTER TABLE IMAGEM ADD(PRIMARY KEY(ID));
```

Figura 5.2 - *Script* da tabela a ser criada em cada *host* da rede

Ao executar o *script* da Figura 5.1 no Editor SQL PLUS pelos diversos *hosts* da rede, uma tabela com o nome `IMAGEM` será criada em cada *host*. Os testes podem ser feitos utilizando a sintaxe do SQL mostrada na Figura 5.2. Considere que o usuário que emitiu a sintaxe esteja localizado no *host* LAB0625 ou no *host* LAB0624.

```
SELECT ID FROM IMAGEM@CARLOS.LAB0623.ORACLE.COM;
```

Figura 5.3 - Instrução SQL para recuperar dados do *host* remoto

Essa instrução SQL resultará na exibição das informações contidas na tabela remota `IMAGEM` localizada no *host* LAB0623 com o nome do Banco de Dados Global `CARLOS.LAB0623.ORACLE.COM`.

5.2.1 Sinônimos (*Synonym*)

Para que o usuário consiga obter a transparência de localização, ele deve criar sinônimos para as tabelas remotas. Os sinônimos são nomes que referenciam objetos por outro nome. Considerando o exemplo mostrado anteriormente, se um sinônimo for criado para a tabela `IMAGEM` do *host* LAB0623, então essa tabela poderá ser acessada como se fosse um objeto do Banco de Dados local.

A sintaxe mostrada na Figura 5.3 é utilizada para a criação de sinônimos.

```
CREATE SYNONYM <NomedoSinônimo>
FOR <objeto_remoto@nome_do_banco_de_dados_global>
```

Figura 5.4 - Sintaxe da SQL para criação de sinônimos

Para criar um sinônimo com nome `TESTE` para a tabela do *script* mostrado na Figura 5.2 o usuário deve executar o comando mostrado na Figura 5.4.

```
CREATE SYNONYM TESTE
FOR IMAGEM@CARLOS.LAB0623.ORACLE.COM;
```

Figura 5.5 - Instrução para criação do sinônimo TESTE.

Após a criação do sinônimo, o usuário poderá utilizar sintaxes de consultas locais para acessar a tabela remota como segue:

```
SELECT ID FROM TESTE;
```

O resultado será uma lista de todas as IDs armazenadas na tabela IMAGEM do *host* LAB0623 que o sinônimo referencia.

5.2.1.1 Sinônimos criados

Neste trabalho a tabela que compõe cada *host* pertencente ao SBDD tem o mesmo nome. Após a criação dos *Links* de Banco de Dados, um sinônimo foi atribuído a cada tabela pertencente a cada máquina, da seguinte maneira:

1. IMAGEM@IMGMED.LAB0625.ORACLE.COM é referenciada pelo sinônimo RAIOX.
2. IMAGEM@IMGMED.LAB0624.ORACLE.COM é referenciada pelo sinônimo ULTRASOM.
3. IMAGEM@CARLOS.LAB0623.ORACLE.COM é referenciada pelo sinônimo TOMOGRAFIA.

5.3 Tipos de dados utilizados

O tipo de dados BLOB e o tipo de dados VARCHAR2, foram os tipos de dados utilizados no projeto.

O tipo de dado BLOB (*Binary Large Object*), é um tipo de dados que possibilita o armazenamento de dados na forma binária em uma coluna de uma tabela. Ele suporta um total de 4 *gigabytes*. O campo BLOB_IMAGEM do *script* da tabela mostrado na Figura 5.1 é utilizado para o armazenamento da imagem médica no Banco de Dados.

O tipo de dados `VARCHAR2` armazena um conjunto de caracteres, com tamanho máximo de 4000 bytes. No *script* mostrado na Figura 5.1 foi criado um campo chamado `URL` que armazenará o *path* do arquivo. Esse campo suporta de 1 a 200 caracteres.

5.4 Conectividade com o Oracle

A linguagem Java acessa um banco de dados por meio do pacote JDBC (*Java Database Connectivity*), que é uma interface padrão para conexão de Java com um Banco de Dados relacional. Esse pacote permite programadores acessar um Banco de Dados para consulta ou alteração utilizando a SQL (*Structured Query Language*) (HORSTMANN e CORNELL, 2000).

Oracle fornece os seguintes *drivers* JDBC para conexão com o Banco de Dados Oracle:

- JDBC *Thin Driver*: é um *driver* que estabelece uma conexão diretamente com o banco de dados, é portável entre as plataformas de SOs, e não necessita de *software* adicional no lado do cliente (HÓLM et al, 2001).
- JDC *OCI Driver*: é um *driver* que faz chamadas a uma API que são construídas em outras linguagens. Requer a instalação do *Oracle Net* no lado do cliente. Não é portável devido às bibliotecas do *driver* serem dependentes do Sistema Operacional (HÓLM et al, 2001).

5.4.1 Estabelecendo a conexão com o Banco de Dados

Como mostrado na Figura 5.5, para a utilização das classes disponíveis pelo Oracle, o usuário deve utilizar o pacote `oracle.jdbc.util` para instanciar objetos de classes do

Oracle. A invocação de uma classe Oracle pode proceder utilizando a instrução `import` seguida pelo nome do pacote, juntamente com o nome da classe a ser utilizada.

Na Figura 5.5 ilustra-se como é feita a conexão com o Banco de Dados, criando uma `String url` que contém os parâmetros necessários para a conexão. O *driver* escolhido para o estabelecimento da conexão com o banco de dados Oracle foi o *Thin driver*, devido aos benefícios que ele oferece.

```
import oracle.jdbc.util.*;
…
 public static Connection getConexao(String host, String port, String
service_name,  String user, String password){ // método que estabelece a
conexão

               String url =
"jdbc:oracle:thin:@"+host+":"+port+":"+service_name;
               Connection con = null ;

               try{
                      Class.forName("oracle.jdbc.driver.OracleDriver");
                      con =
DriverManager.getConnection(url,user,password);
               }catch (SQLException sqlEx){   …
               }catch ( ClassNotFoundException classEx){   …
               }finally{ …
               }
       }
```

Figura 5.6 - Trecho de código que estabelece a conexão com o BD Oracle.

Como mostrado na Figura 5.5, a conexão é feita passando o nome do *host* (computador ao qual o usuário está conectado), número da porta (número da porta que o *Listener* está esperando por conexões), o nome do Serviço (Nome do Banco de Dados local), o usuário e a senha para o método estático `getConexao()`, que por sua vez passará as informações ao método `getConnection()`, também estático, da classe `DriverManager`.

O método `forName()` tenta carregar o *driver* passado como parâmetro. Caso aconteça alguma anomalia no carregamento do mesmo, uma exceção do tipo `ClassNotFoundException` será lançada.

O método estático `getConnection()` da classe `DriverManager` tentará estabelecer a conexão e retornará um objeto do tipo `Connection` caso a conexão seja bem sucedida, caso contrário o método lançará uma exceção do tipo `SQLException`.

A instância do objeto `Connection` retornado por `getConnection()` será atribuído à referência `conn` do tipo `Connection`. O usuário informado como parâmetro para o método `getConnection()` deverá estar criado no banco de dados e ter permissão de acesso ao Banco de Dados.

5.5 Armazenamento dos dados

Nessa etapa da implementação foram desenvolvidos dois programas para fazer a inserção dos dados nas tabelas: o primeiro executa o armazenamento da imagem utilizando o campo do tipo `BLOB`. Essa inserção é feita tanto na tabela local quanto nas tabelas remotas. O segundo programa consiste em inserir as informações referentes ao *path* do arquivo utilizando o tipo de dados `VARCHAR2`. Algumas considerações devem ser salientadas a respeito desse *software*, pois o armazenamento do *path* remoto implicará na transferência da imagem para o *host* remoto.

O processo de armazenamento consiste em popular a tabela local como também as tabelas remotas e obter os tempos de duração do processo. É inserida uma determinada quantidade de registros fornecida pelo usuário. Essa quantidade é dividida em partes iguais com base em um critério estipulado no início do processo. Esse critério pode ser observado nos anexos E, F, G, e H.

5.5.1 Armazenando o tipo BLOB

Os parâmetros do método `load()` da classe `OperacoesTCC,` pode ser observados na Figura 5.6 que apresenta a declaração do método `load()`.

```
public int load( String enderArq, int quantArm, int idCorrente, String
tabela)
```

Figura 5.7 - Declaração do método load().

Quando o programa invoca o método `load()`, será executado um *loop* que irá inserir o número de registros informado na tabela. Uma instrução PL/SQL foi construída para retornar uma referência do campo `BLOB_IMAGEM` para que fosse possível o armazenamento da representação binária da imagem. A instrução PL/SQL é apresentada na Figura 5.7.

```
Begin INSERT INTO "+ tabela +"(ID,BLOB_IMAGEM) VALUES (? ,empty_blob())
" +  " return BLOB_IMAGEM into ?; " + " end;"
```

Figura 5.8 - Instrução PL/SQL usada para inserir os dados no Banco de Dados.

Um objeto do tipo `FileInputStrem` é instanciado para que os dados na forma binária sejam lidos da imagem. O parâmetro passado para o construtor da classe `FileInputStream` é o *path* físico do arquivo, que é um parâmetro do método `load()`. Na seqüência a instrução SQL é preparada e os parâmetros da mesma são configurados. O método `registerOutParameter()` registra o segundo parâmetro da instrução PL/SQL mostrado na Figura 5.8 como sendo do tipo `BLOB` da classe `OracleTypes`.

```
…
        OracleCallableStatement ocs1 = (OracleCallableStatement)
conn.prepareCall(sql1);
        ocs1.setInt(1,id);  /* atribui o valor da variável id ao
primeiro parâmetro da instrução SQL */
        ocs1.registerOutParameter(2,OracleTypes.BLOB); /* registra o
Segundo parêmetro como BLOB*/
        ocs1.executeUpdate();
…
```

Figura 5.9 - Trecho preparação e execução da instrução PL/SQL.

O objeto `ocs1` do tipo `OracleCallableStatement`, exibido na Figura 5.8, é criado para executar a instrução PL/SQL e armazenar o valor de retorno, que será recuperado mais tarde com o método `getBlob()`. O primeiro parâmetro da instrução PL/SQL é um parâmetro do tipo inteiro que utiliza o método `setInt()` da classe `OracleCallableStatement` para registrar o seu valor. Já o segundo parâmetro é um parâmetro de retorno que retornará um objeto do tipo `BLOB`. O método `registerOutParameter()` registra o parâmetro de saída para ser do tipo `OracleTypes.BLOB`. Em seguida a instrução PL/SQL é executada pelo método `executeUpdate()`. A instrução PL/SQL foi passada como parâmetro do método `prepareCall()` da classe `Connection` que criará um objeto com a instrução PL/SQL a ser preparada para a execução.

Na Figura 5.9 mostra-se o trecho de código utilizado para inserir a imagem no campo do tipo `BLOB`. O método `getBlob()` é utilizado para obter o valor retornado pela PSL/SQL e um objeto do tipo `BLOB` é instanciado. O método `getBinaryOutputStream()` retorna um objeto do tipo `OutputStream` que possibilita a escrita da imagem no campo `BLOB_IMAGEM` da tabela mostrada no *script* da Figura 5.1. O restante do código apresentado pela Figura 5.9 lê os *bytes* da *stream* de entrada (`FileInputStream`) e escreve os *bytes* lidos na *stream* de saída (`OutputStream`).

```
...
    BLOB blob = ocs1.getBLOB(2);

    OutputStream os = blob.getBinaryOutputStream(); //stream de saída
    int size = blob.getBufferSize();
    byte buffer1[] = new byte[size];
    int length;

    wcount = 0;
    while(( length = is.read(buffer1,0,size)) != -1){ //lendo do arquivo
          wcount += length;
          os.write(buffer1,0,length); //escrevendo no campo BLOB_IMAGEM
    }
...
```

Figura 5.10 - Trecho de código que armazena a imagem no campo BLOB_IMAGEM da tabela local

Os trechos de códigos mostrados anteriormente são utilizados para a inserção da representação binária da imagem no campo `BLOB_IMAGEM` da tabela local.

Ao fazer o teste com esse programa para a inserção em tabelas remotas a exceção mostrada pela Figura 5.10 foi lançada pelo método `executeUpdate()`.

"ORA-22992 – localizadores LOB não podem ser acessados em tabelas remotas"

Figura 5.11 - Exceção lançada ao tentar inserir uma imagem em uma tabela remota

Estudos foram feitos para tentar solucionar o problema de forma que o programa pudesse armazenar a imagem tanto local quanto remota. Foi encontrada na documentação do Oracle, uma restrição do Oracle9i que informa que tipos de dados LOBs (*Large Object*) não são suportados de forma distribuída.

Uma solução para o problema apresentado foi a criação de uma tabela auxiliar idêntica à tabela `IMAGEM` apresentada na Figura 5.1. Na Figura 5.11 exibe-se a criação da tabela `AUXILIAR`.

```
...
String sql2 = "CREATE TABLE AUXILIAR AS (SELECT MAX(ID) FROM IMAGEM
)";


OracleStatement stm = (OracleStatement)conn.createStatement();
stm.executeUpdate(sql2);
...
```

Figura 5.12 - Criação da tabela AUXILIAR em tempo de execução.

Após a criação da tabela auxiliar é feita a inserção somente do campo `ID` na tabela remota. Os dados da imagem são lidos por um trecho de código idêntico ao mostrado na Figura 5.9 e escritos no campo `BLOB_IMAGEM` da tabela `AUXILIAR`. A inserção desse campo na tabela remota é feita através de uma instrução `UPDATE` da SQL. Na Figura 5.12 exibe-se como é feito esse processo.

```
...
   String sql4 = "UPDATE "+tabela+" SET BLOB_IMAGEM = (SELECT
BLOB_IMAGEM FROM AUXILIAR WHERE ID = "+id+" ) WHERE ID = "+id;

   OracleStatement ostm = (OracleStatement) conn.createStatement();
   ostm.executeUpdate(sql4);
...
```

Figura 5.13 - Trecho de código para inserção no campo BLOB_IMAGEM da tabela remota.

A inserção na tabela remota é feita utilizando o comando `UPDATE` em conjunto com uma *Sub-Query*, que retorna o valor do campo `BLOB_IMAGEM` da tabela `AUXILIAR`, selecionando apenas o registro corrente na cláusula `WHERE`.

Após esse processo a tabela `AUXILIAR` é removida e a inserção na tabela remota é finalizada com a execução do método `commit()` que valida as atualizações de DML. Como mostrado na Figura 5.13.

```
...
sql4 = "DROP TABLE AUXILIAR";
ostm.executeUpdate(sql4);
...
```

Figura 5.14 - Remoção da tabela AUXILIAR em tempo de execução.

5.5.2 Armazenando o *path* da imagem

No processo de armazenamento do *path* alguns cuidados devem ser tomados. Quando o *path* de uma imagem é armazenado em uma tabela remota, a imagem que é referenciada pelo *path* deverá ser transmitida para o *host* remoto e armazenada na mesma hierarquia de diretórios como armazenada no atributo `URL` da tabela. Portanto, o mecanismo de *socket* foi utilizado para fazer a transferência da imagem do *host* local para a máquina que contém a tabela que está sendo utilizada na inserção dos dados.

O processo de transferência do arquivo do *host* local para o *host* remoto é feito utilizando um servidor *socket*, que deverá ser inicializado manualmente em cada máquina antes de iniciar o processo de inserção, e um cliente *socket*, que será instanciado pelo método `loadPath`.

O cliente é instanciado quando o construtor da classe `ClientArm` é executado, como mostra o trecho no final do código apresentado na Figura 5.14. O construtor do *socket* cliente tem como parâmetro o nome do *host* ao qual ele deverá conectar e o endereço físico do arquivo que deseja transferir.

Ao ser invocado o método `runClient()` da classe `ClientArm`, inicia-se o processo de leitura do arquivo no *host* local e a transferência do mesmo para o *host* remoto. O código fonte do *socket* Cliente (`ClientArm`) e do *socket* Servidor (`ServerArm`) podem ser observados com mais detalhes nos ANEXOS I e J, respectivamente.

O início do trecho de código apresentado na Figura 5.14 mostra as configurações e métodos necessários para a inserção do *path* da imagem na tabela local ou remota informada no parâmetro `tabela`. O parâmetro `idtabela` do método `loadPath()` é utilizado para a escolha da tabela a ser utilizada. Se o valor informado nesse parâmetro for o valor 0 (zero), então a tabela utilizada será a tabela local. No caso se a tabela selecionada não for à tabela local (`idtabela` diferente de zero) a instrução `switch` será executada para identificar o nome do *host* para onde a imagem deverá ser transferida.

O nome do *host* é um parâmetro do construtor da classe `ClientArm` que ao ser instanciado e executado seu método `runClient()`, conectará ao servidor *socket* remoto localizado no *host* passado como parâmetro. O método `runClient()` deve ser invocado para dar inicio ao processo de transferência.

O laço apresentado no método `loadPath()`, mostrado na Figura 5.14, insere um valor total de registros informado pelo parâmetro `quantArm`. A instrução SQL utilizada na inserção na tabela local quanto na tabela remota é a mesma. A instrução SQL é exibida na Figura 5.14 dentro do laço de repetição.

Na inserção do *path* da imagem na tabela remota, um objeto `client` do tipo `ClientArm` é instanciado para estabelecer uma conexão com o servidor *socket* (`ServerArm`). O *host* que instancia o Cliente deve executar o método `runClient()` fazendo com que o cliente localize o servidor.

```
...
/* método para a inserção do path da imagem */
public int loadPath(String enderArq, int quantArm, int idCorrente,
String tabela , int idtabela){
       ...
       if (idtabela==0){ /*Executado se for a tabela local*/
              for (id=idCorrente+1; id <= quantArm+idCorrente; id++){
                     String sql = "INSERT INTO "+tabela+" (ID,URL)
VALUES ("+id+",'"+enderArq+"')";

                     OracleStatement stm = (OracleStatement)
conn.createStatement();
                     stm.executeUpdate(sql);
                     conn.commit();
                     stm.close();
              }
       }else{
              switch(idtabela){ /* identificação do host remoto */
                     case 1: host = "LAB0624";break;
                     case 2: host = "LAB0623";break;
              }
              for (id=idCorrente+1; id <= quantArm+idCorrente; id++){
                    String sql = "INSERT INTO "+tabela+" (ID,URL)
       VALUES ("+id+",'"+enderArq+"')"; /* Instrução SQL para a
       inserção dos dados */
              OracleStatement stm = (OracleStatement)
conn.createStatement();
                     stm.executeUpdate(sql);
                     conn.commit();
                     stm.close();

       /* instanciacao do cliente */

       ClientArm client= new ClientArm(host,enderArq);
       client.runClient(); /* invocação do método que inicia a
transferência do arquivo */
              }
       }
 ...
```

Figura 5.15 - Trecho de código para a inserção do *path*

5.6 Recuperação dos dados

Para a recuperação dos dados, dois programas foram desenvolvidos: um programa para recuperar a imagem armazenada em sua forma binária no campo do tipo `BLOB`, tanto na tabela local quanto nas tabelas remotas e o outro para recuperar o *path* da imagem armazenado no campo `URL` da tabela mostrada na Figura 5.1. Na recuperação do *path* da

imagem, deve ser considerado que ao recuperar o *path* o software ficará responsável em executar a transferência do arquivo para o *host* que executou a consulta.

O processo de recuperação consiste em recuperar os registros existentes na tabela local ou em alguma das tabelas remotas, obtendo o tempo de recuperação ao final do processo.

Um critério apresentado no início do programa determina o número total de tabelas que deverão ser consultadas. Esse critério tem como base o número total de registros informado pelo usuário ao iniciar o programa. Este critério pode ser observado no código do ANEXO E.

5.6.1 Recuperando o tipo BLOB

Na recuperação da representação binária da imagem, o registro é recuperado com uma instrução `SELECT` simples que seleciona apenas a `ID` e o atributo `BLOB_IMAGEM`. Após a recuperação do objeto do campo `BLOB_IMAGEM`, os dados contidos nesse campo são escritos em um arquivo do tipo TIFF. A recuperação do tipo `BLOB` em tabelas remotas apresenta o mesmo problema descrito na seção anterior, sendo que procedimentos adicionais foram implementados para selecionar o campo `BLOB_IMAGEM` da tabela remota.

O processo de recuperação inicia quando o método `save()` é chamado. Esse método possui dois parâmetros: o primeiro indica se a tabela é local ou remota e o segundo indica o nome da tabela a ser utilizada. Na Figura 5.15 ilustra-se a declaração do método `save()`.

```
...
public int save( int idtabela , String tabela)
...
```

Figura 5.16 - Declaração do método save().

Ao ser invocado o método `save()`, este invocará o método estático `getConexao()` da classe `Conexão` que estabelecerá a conexão com o Banco de Dados e retornará um objeto do tipo `Connection`. Em seguida criará uma instrução SQL simples para obter os dados da tabela cujo nome foi o fornecido como argumento desse método. Na Figura 5.16 mostra-se a declaração da instrução de consulta.

```
...
   sql = "SELECT ID,BLOB_IMAGEM FROM "+ tabela;
...
```

Figura 5.17 - Instrução SQL para recuperação dos valores dos campos ID e BLOB_IMAGEM.

Caso a tabela seja local, o método `executeQuery()` é executado para obter as informações solicitadas pela instrução da Figura 5.16.

Após a recuperação dos dados do Banco de Dados, o resultado é atribuído a uma referência do tipo `OracleResultSe`t. Caso o objeto não contenha nenhuma linha, um erro será emitido informando que nenhum registro foi encontrado. Caso contrário o trecho de código mostrado na Figura 5.17 será executado.

O trecho de código da Figura 5.17 armazena a imagem contida no campo `BLOB_IMAGEM` da tabela consultada em um arquivo que terá o nome inicial de "`Fig`" concatenado com o valor da chave primária do registro (`ID`), concatenada com a extensão `.Tiff`. O software desenvolvido não trata tipo e nem nomes das imagens armazenadas como

também não trata o local onde deve ser armazenado, armazenando a imagem na pasta corrente onde o *software* está sendo executado.

```
...
    do {
         fileName = "Fig"+ors.getInt("ID")+".Tiff";
         BLOB blob = ors.getBLOB(2);
         InputStream is = blob.getBinaryStream();
         FileOutputStream fos = new FileOutputStream( fileName );

         int size = blob.getBufferSize();
         byte buffer1[] = new byte[size];
         long wcount = 0;
         int length;

         while ((length = is.read(buffer1, 0, size)) != -1) {
              wcount += length;
              fos.write(buffer1,0,length);
         }

      ...

      }while(ors.next());
...
```

Figura 5.18 - Código que armazena a imagem contida no campo BLOB_IMAGEM no hd local.

Na Figura 5.17 mostra-se a leitura do campo do tipo `BLOB` e a escrita do mesmo para um arquivo externo. O arquivo é criado quando a instrução `FileOutputStream fos = new FileOutputStream(fileName)` é executada. O arquivo será criado com o nome fornecido como parâmetro do mesmo. O código restante lê do campo `BLOB_IMAGEM` uma seqüência de *bytes* e escreve no arquivo, ao final do processo a imagem estará armazenada no diretório corrente da aplicação.

Ao tentar recuperar o tipo `BLOB` em tabelas remotas, a mesma exceção apresentada na Figura 5.10 foi lançada pelo método `executeQuery()`.Por isso algumas alterações necessitaram ser feita no *software*.

Para conseguir a recuperação da imagem contida no campo `BLOB` da tabela remota, uma tabela temporária com o nome `AUXILIAR`, deverá ser criada no *host* local. Essa tabela é criada contendo todos os valores da tabela remota. Isso é possível devido a uma instrução de

seleção associada com a instrução de atualização. Na Figura 5.18 ilustra-se a instrução de criação da tabela AUXILIAR.

```
...
sql = "SELECT ID,BLOB_IMAGEM FROM "+ tabela ;
...

sql1 = "CREATE TABLE AUXILIAR AS (" + sql + ")";
```

Figura 5.19 - Instrução de criação da tabela AUXILIAR para recuperação do campo BLOB_IMAGEM

Com isso, as informações ficam localizadas no *host* local. Deverá ser emitida uma consulta para buscar os dados na tabela AUXILIAR (Figura 5.19) e o processo de recuperação no *host* local será executado como mostrado anteriormente.

```
String sql2 = "SELECT ID,BLOB FROM AUXILIAR";
```

Figura 5.20 - Instrução de seleção dos dados na tabela AUXILIAR

5.6.2 Recuperação pelo *path*

Alguns cuidados devem ser tomados pelo DBA na recuperação do arquivo pelo *path*. Ao recuperar o *path* do arquivo no banco de dados, a imagem deve ser também recuperada para o *host* local.

O *software* que recupera o *path* da imagem foi desenvolvido utilizando o mecanismo de *socket* para trazer a imagem do *host* remoto para o *host* local. Devido ao fato de que o *software* não armazena a imagem na estrutura de diretório adequada, o usuário deve garantir que a imagem a ser recuperada encontra-se no diretório que o *path* armazenado indica. O

servidor deverá ser iniciado manualmente em cada máquina participante do SBDD antes de dar início à transação.

Na Figura 5.20 exibe-se a declaração do método `savePath()`, que é chamado quando deseja-se recuperar uma imagem do Banco de Dados pelo seu *path*.

```
...
public int savePath(int idtabela, String tabela)
...
```

Figura 5.21 - Declaração do método savePath().

Ao ser invocado o método `savePath()`, o parâmetro *idtabela* é verificado para saber qual o nome do *host* que contém a tabela. Esse nome é armazenado em uma variável local para posteriormente ser usado na instância do cliente. Na Figura 5.21 mostra-se esse processo.

```
...
switch(idtabela){ /* Seleção da tabela local ou remota */
        case 0: host = localHost;
        case 1: host = "LAB0624";
        case 2: host = "LAB0625";
 }
 ...
```

Figura 5.22 - Seleção do *host* a ser utilizado na instância do cliente

Na Figura 5.22 mostra-se a instrução SQL para recuperar os dados da tabela. Os registros selecionados serão armazenadas em um objeto do tipo `OracleResultSet` que tem nome de `ors` no trecho de código exibido na Figura 5.22. A Figura 5.22 mostra-se ainda o objeto responsável pela execução do comando.

```
…
   sql = "SELECT ID,PATH FORM "+tabela;
…
   Connection conn =
   Conexao.getConexao(localHost,porta,servico,user,senha); /*conecta ao
   BD */
   conn.setAutoCommit(false);
   OracleStatement os = (OracleStatement) conn.createStatement();
   OracleResultSet ors = (OracleResultSet)os.executeQuery(sql); /*
Executa a instrução sql contida na variável sql*/
…
```

Figura 5.23 - Instrução SQL para seleção do path e execução da mesma.

O comando `while` mostrado na Figura 5.23 será executado enquanto houver registros no objeto `ors`. Em cada iteração do comando `while` um objeto do tipo `Client` é instanciado. Os parâmetros passados para o construtor da Classe `Client`, são o nome do *host*, selecionado logo no início do método `savePath()`, o *path* da imagem, e o *id* da imagem. Os dois últimos serão utilizados para compor o nome do arquivo.

```
...
   while(!ors.next()){
      Client client = new Client(host,ors.getString(2),
ors.getInt(1));
      client.runClient();
      cont++;
   }
...
```

Figura 5.24 – Trecho de código que instancia o Cliente.

A iteração entre o cliente e o servidor *socket* ocorre da seguinte forma:

1. O Cliente é instanciado e envia o *path* do registro corrente para o servidor.
2. O Servidor lê o arquivo no *path* informado e envia o mesmo para o cliente.
3. O cliente recebe o arquivo em forma de *stream* e cria o arquivo na máquina onde está sendo executado.

4. O cliente é finalizado e o servidor continua ativo na espera por outras conexões.

Detalhes do código do cliente e do servidor podem ser encontrados nos ANEXOS C e D, respectivamente.

6 - RESULTADOS E DISCUSSÕES

Após a fase de instalação do SGBD Oracle Distribuído e da implementação dos *softwares*, iniciou-se a fase de testes com a finalidade de atingir os objetivos propostos por esse trabalho.

Nesta etapa foram utilizadas três máquinas *Pentium* IV 2.8 *Gigahertz* com 512 *Megabytes* de memória *RAM* e 80 *Gigabytes* de *HD*.

Os procedimentos adotados para a execução dos testes foram baseados em quatro critérios: tamanho da imagem a ser inserida, número de máquinas pertencentes ao sistema, condições para inserção e recuperação, e os dois tipos de dados utilizados. Os testes apresentados nesse capítulo foram executados apenas na máquina LAB0625 com a utilização de uma imagem médica de 5 *MB*.

Em relação ao critério do tamanho das imagens, foi proposta a inserção de imagens com tamanho de 5 *MB*, 10 *MB* e 15 *MB*, utilizando três máquinas. Quanto ao critério de inserção e recuperação, foi estipulada a divisão de uma quantidade variável de registros, em duas ou três partes de tamanhos iguais.

Os testes apresentados nesse capítulo têm como base o critério de no máximo 50 registros a serem inseridos ou recuperados na máquina local. Se o número de registro informado for maior que 50 e menor que 101, os registros serão divididos em duas partes iguais e será armazenada uma parte local e a outra parte em um dos sites remotos. Caso o valor informado seja superior a 100, os registros serão divididos em três partes iguais e serão inseridos nas três máquinas participantes do sistema. Uma imagem médica de 5 *MB* (ANEXO L) foi a imagem empregada nos testes.

Este fase está dividida em duas etapas:

1. Testes de armazenamento dos dados.
2. Testes de recuperação dos dados.

Para as duas fases foi medido o tempo de atraso que a rede proporciona ao executar uma operação remota. Após a execução dos testes, as tabelas eram deixadas vazias como se encontravam no início da operação.

6.1 Armazenamento da imagem

Como já mencionado, os testes de armazenamento consistiram na inserção de um valor variável de registros pelos diversos *hosts* participantes do sistema. O armazenando foi executado das duas formas: no campo `BLOB_IMAGEM` e usando o *path* da imagem.

Foram feitos alguns testes com o campo do tipo `BLOB`, na qual foi possível fazer comparações quanto ao armazenamento da imagem no *host* local e quanto ao armazenamento da imagem nos *hosts* remotos. As comparações feitas com o tipo de dado `BLOB` foram também feitas usando o *path* da imagem. Após as discussões sobre o armazenamento da representação binária e do *path* da imagem, foi executada a comparação do armazenamento entre os dois tipos de dados.

O armazenamento da representação binária mostrou melhor desempenho quando almeja armazenar dados em tabelas remotas. Já o *path* da imagem mostrou tempos longos no armazenamento remoto e mostrou ser eficiente no armazenamento local.

O gráfico da Figura 6.1 exibem-se os tempos obtidos no teste de armazenamento da representação binária da imagem na tabela local e nas tabelas remotas do SBDD.

A tabela mostrada junto com o gráfico exibe os tempos obtidos quando se deseja armazenar 1, 40 e 100 registros nas tabelas participantes do sistema. As tabela `RAIOX` é a tabela localizada no *host* local, e as demais localizadas nos *hosts* remotos.

A análise do gráfico apresenta que na inserção de um único registro em qualquer dos *hosts* remotos, o tempo será o mesmo. Ao fazer a mesma comparação com um número maior de registros, pode ser observado no gráfico que os tempos de inserção entre as tabelas remotas tende a divergir.

Com relação ao armazenamento da representação da imagem devem ser levados em consideração fatores como a presença da rede de comunicação, citada anteriormente, o fato de se utilizar uma tabela auxiliar para executar o armazenamento da imagem no campo do tipo `BLOB` de uma tabela remota e o fato da linguagem de pro como foi apresentado na Seção 5.4.1, Capítulo 5.

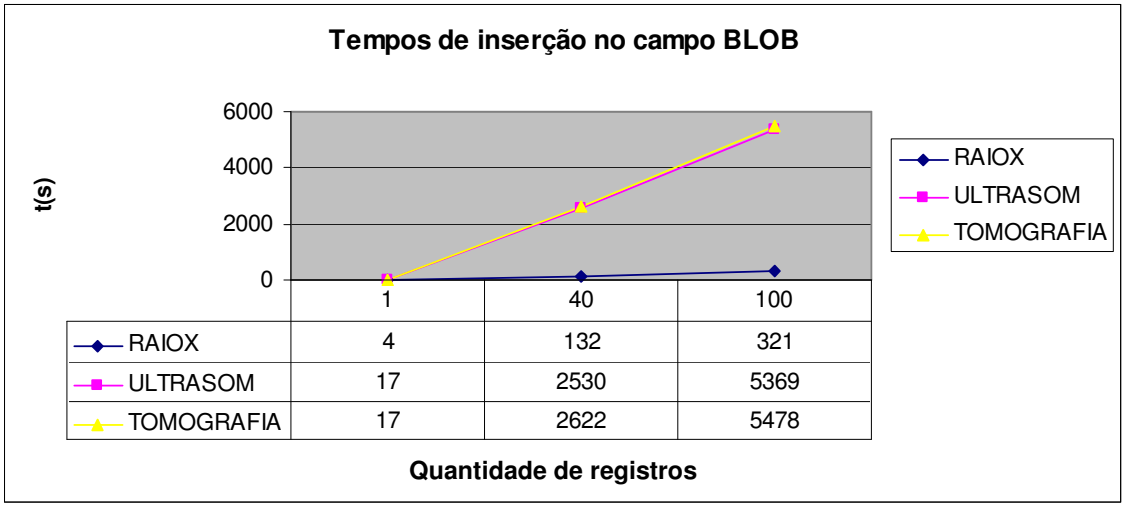


Figura 6.1 - Comparação de inserção local e remota utilizando o tipo de dado BLOB

Ao armazenar um único registro o tempo é superior ao tempo médio obtido quando se deseja armazenar um número maior de registros na mesma tabela. Essa comparação pode ser obtida através da razão do tempo obtido pelo o número total de registros inseridos. Portanto ao aumentar o número de registros o tempo médio de inserção em cada tabela, tende a diminuir.

O gráfico exibido na Figura 6.2 exibe o armazenamento do *path* da imagem na tabela local e nas tabelas remotas. Os tempos obtidos nesse processo foram superiores aos tempos apresentados no armazenamento da representação binária da imagem.

O gráfico apresentado na Figura 6.2 exibe os tempos obtidos no processo de armazenamento do *path* da imagem. A tabela disponibilizada na parte inferior do gráfico exibe os tempos obtidos para cada quantidade de registros inseridos (1, 40 e 100) nas tabelas do sistema. As tabelas `ULTRASOM` e `TOMOGRÁFIA` estão localizadas nos computadores remotos nos quais foi feito o teste, e a `RAIOX` é a tabela local.

Do gráfico apresentado pela Figura 6.2 nota-se que o tempo apresentado na inserção do *path* da imagem no *host* local é inferior ao tempo de inserção do mesmo no Banco de Dados remoto. Isto ocorre devido à presença da rede e a necessidade de transferência da imagem do *host* local para o *host* remoto.

Os tempos apresentados na inserção do *path* da imagem nos dois *hosts* remotos também divergem quando aumenta o número de registros.

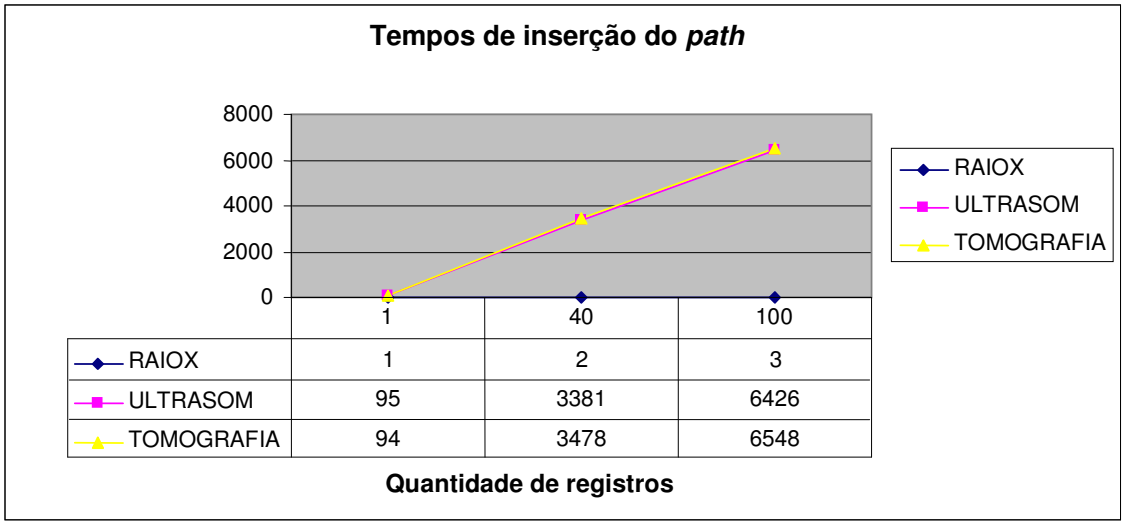


Figura 6.2 - Comparação da inserção local e remota utilizando o *path* da imagem

Nesse processo o tempo médio na inserção de um número maior de registros também tende a diminuir com relação à inserção de um menor número de registros.

A Figura 6.3 exibe-se o gráfico de comparação entre o armazenamento dos diferentes tipos de dados na tabela local. A inserção da imagem no Banco de Dados local é um processo mais lento, pois para o armazenamento da imagem no Banco de Dados é necessária a leitura binária da mesma, e logo em seguida a escrita binária da mesma no campo `BLOB_IMAGEM` da tabela local. Já na inserção do *path*, a transferência da imagem não é necessária.

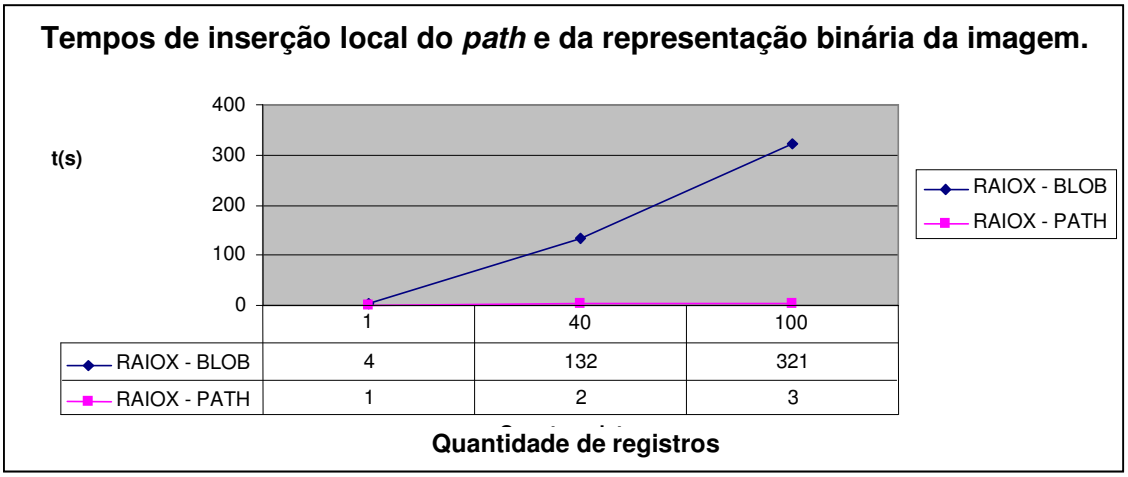


Figura 6.3 - Comparação entre a inserção do path da imagem, e a inserção da imagem no banco de dados

Os tempos apresentados na inserção da representação binária da imagem em relação ao armazenamento do *path* é mais lento quando deseja executar o armazenamento em *hosts* remotos. Já a execução no *host* local, o path demonstra ser mais ágil. Esse resultado pode ser observado no gráfico da Figura 6.3.

O próximo gráfico apresentado na Figura 6.4 mostra a comparação dos diferentes tipos de dados sendo armazenados nas tabelas remotas. A tabela inferior apresentada no gráfico demonstra os tempos obtidos na inserção das diferentes quantidades (1, 40 e 100) registros nas tabelas `ULTRASOM` e `TOMOGRAFIA` que estão localizadas nos *hosts* remotos.

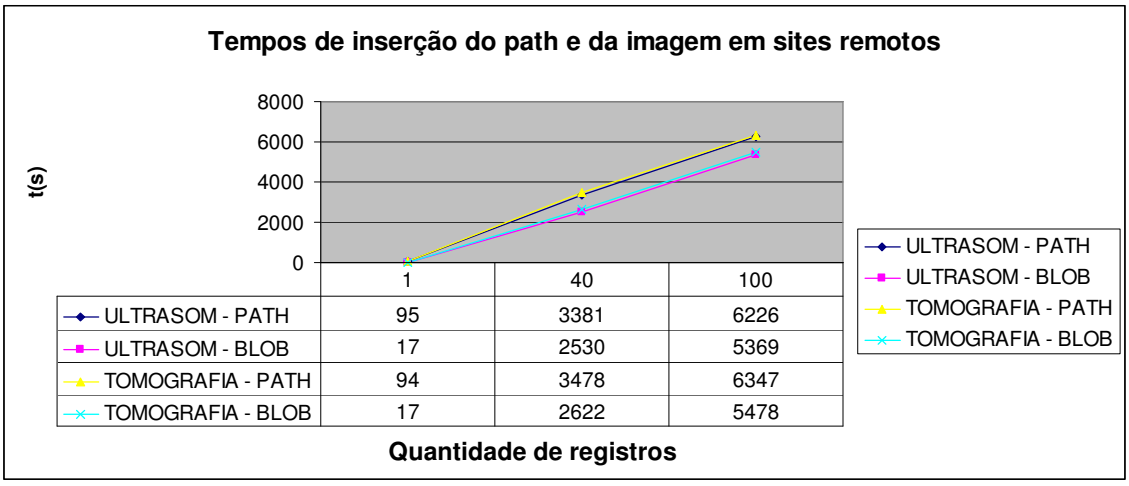


Figura 6.4 - Comparação entre os tipos de dados na inserção remota

A partir do gráfico apresentado na Figura 6.4 observa-se que os tempos para inserção da imagem no campo `BLOB_IMAGEM` da tabela remota, resulta em tempos inferiores aos tempos de inserção do *path* que deve transferir a imagem para o *host* remoto. O gráfico apresenta também que a inserção da imagem na tabela `ULTRASOM`, utilizando o tipo de dados `BLOB`, é mais eficiente que a mesma inserção na tabela `TOMOGRAFIA`, como também pode ser observado que o armazenamento do *path* da imagem na tabela `ULTRASOM` é mais ágil que a mesma inserção na tabela `TOMOGRAFIA`. Essas diferenças podem ser resultantes do estado de processamento, utilização da memória *RAM* (*Random Access Memory*), espaço em disco disponível no momento dos testes e os atrasos que a rede proporciona.

Com relação aos testes apresentados para a inserção de dados, conclui-se que a utilização do armazenamento do *path* da imagem é mais eficiente quando se deseja manter a imagem no *host* local devido não ser necessária a transferência da imagem. A utilização do armazenamento da representação binária da imagem nos bancos de dados remotos apresentou uma melhor performance sobre os teste de armazenamento do *path* da imagem. A justificativa

pelos altos tempos apresentados na inserção do *path* deve-se ao fato de ser necessária a transferência da imagem do *host* local para o *host* remoto.

Dos testes apresentados conclui-se que o armazenamento da representação binária da imagem é mais rápido e confiável para sistemas que necessitam de armazenar imagens médicas em *hosts* remotos. Para o armazenamento no *host* local, a utilização do armazenamento do *path* apresentou melhor desempenho, mas não tão seguro quanto o armazenamento da representação binária da imagem. A segurança refere-se à possível inconsistência que pode ocorrer quando se armazena somente o *path* da imagem. No caso da imagem ser apagada, levará a um estado de inconsistência de dados. Esse fato já não ocorre para o armazenamento da representação binária da imagem devido à mesma estar armazenada no banco de dados.

Dos testes apresentados pode-se conclui que quanto maior o número de registros a ser inserido o tempo médio de inserção dos registros, em cada tabela, tende a diminuir. Isto ocorre devido ao fato da linguagem Java necessitar de instanciar objetos.

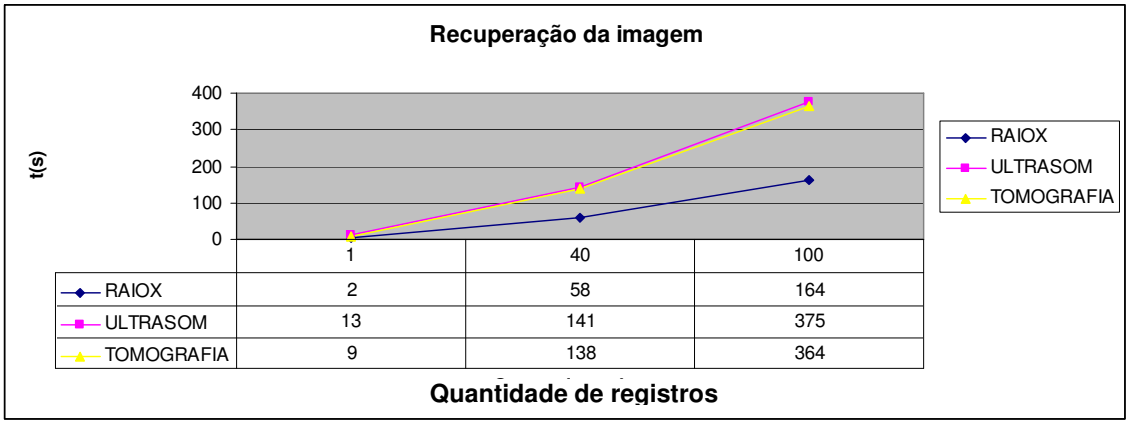
6.2 Recuperação da imagem

O gráfico apresentado pela Figura 6.6 exibe a recuperação da representação binária da imagem na tabela RAIOX no *host* local, como também nas tabelas ULTRASOM e TOMOGRAFIA dos *hosts* remotos.

Os tempos obtidos são baseados em 3 (três) classes de registros, na qual cada classe apresenta uma quantidade de registro a ser recuperada. Na primeira classe é recuperado apenas um registro, na segunda classe são recuperados 40 registros. Na terceira e última classe são recuperados 100 registros.

Na Figura 6.5 exibe-se o gráfico com os tempos obtidos na recuperação da imagem utilizando o *path* da mesma armazenado no banco de dados, pelos três *hosts* participantes do

sistema. A tabela `RAIOX` encontra-se no host local enquanto a tabela `ULTRASOM` e `TOMOGRAFIA` encontram-se nos *hosts* remotos.



	1	40	100
RAIOX	2	58	164
ULTRASOM	13	141	375
TOMOGRAFIA	9	138	364

Figura 6.5 - Comparação entre os tempos local e remoto.

Os resultados obtidos, apresentados pela Figura 6.6, mostram que entre as duas tabelas remotas participantes do sistema, a tabela `TOMOGRAFIA` apresentou um melhor desempenho para a recuperação dos três conjuntos de registros. Isto deve-se ao estado de processamento, utilização da memória *RAM*, espaço em disco disponível e aos atrasos que a rede proporciona.

A recuperação dos registros no *host* local apresentou tempos inferiores aos tempos de inserção da imagem. Considerando que o processo de recuperação da representação binária da imagem deve ler a imagem no campo BLOB e deve armazená-la no *host* solicitante, como também deve ser levada em consideração a necessidade da utilização de uma tabela auxiliar para a recuperação da imagem.

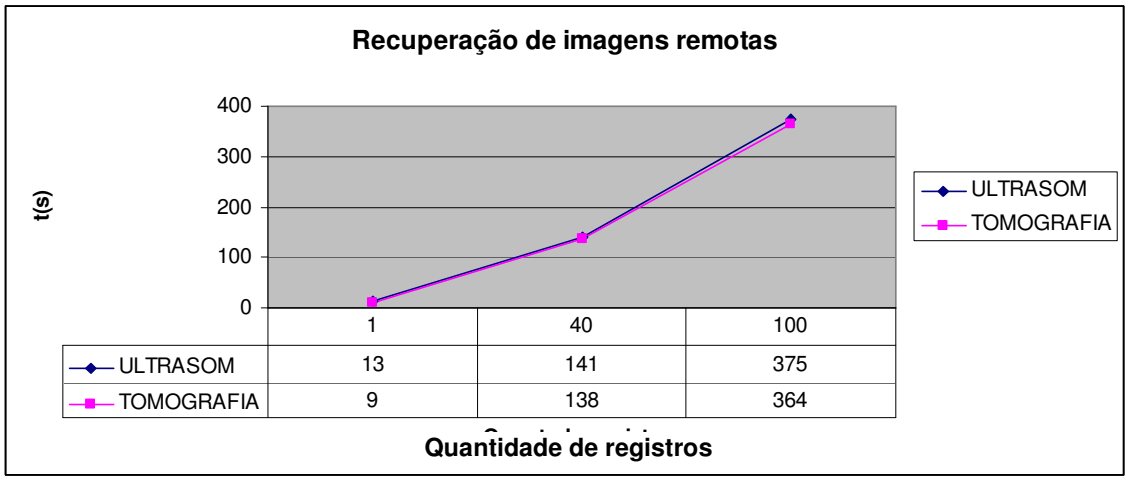


Figura 6.6 - Comparação da recuperação da imagem em *hosts* remotos

Com base nos dados obtidos no armazenamento e na recuperação da representação binária da imagem nas tabelas tanto local quanto remota, conclui-se que a operação de recuperação tem melhor desempenho sobre a operação de armazenamento da representação binária da imagem. Em relação às tabelas remotas pode-se concluir que a recuperação utilizando a tabela auxiliar, como mostrada na Seção 5.5.1, Capítulo 5, pode influenciar nos tempos obtidos.

Nessa etapa não foi possível obter uma conclusão entre a recuperação do *path* da imagem e a recuperação da representação binária da imagem, devido a um erro encontrado ao tentar recuperar o *path* da imagem nos *hosts* remotos. Tal erro apresentado é um erro interno do Oracle que foi lançado pelo método `executeQuery()` durante a execução do programa de recuperação do *path* da imagem. O erro é apresentado na Figura 6.7.

Pesquisas foram realizadas para tentar solucionar o problema, mas não foi possível resolver o mesmo. Alguns fóruns na Internet consideram este erro como um *bug* do Oracle e outros consideram um erro no *CLASSPATH* configurado pela linguagem de programação Java e outro configurado pelo Oracle.

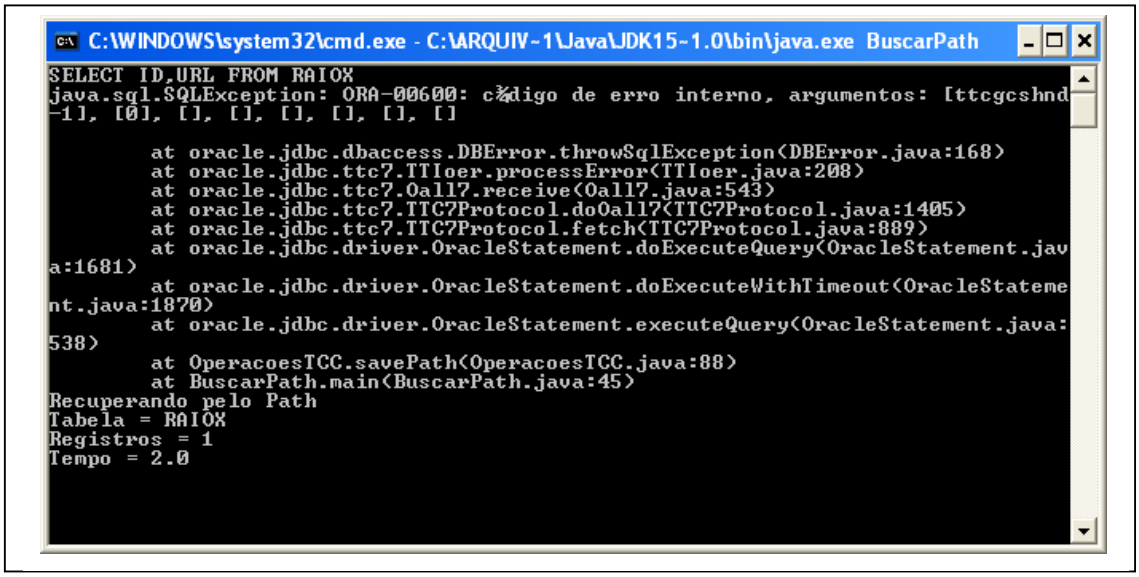


Figura 6.7 - Erro apresentado na recuperação do path em tabelas remotas

6.3 Considerações Finais

Ao ser estudada a documentação do Oracle, uma possível solução apareceu para solucionar a restrição apresentada pelos tipos de dados LOBs. Tal solução seria a utilização da replicação avançada.

Utilizando a replicação avançada e fazendo com que cada *host* participante do SBDD contenha um *snapshot* (Seção 4.2) completo da tabela original de cada *host*, e um período de tempo atribuído a esse *snapshot*. Os dados serão armazenados no *snapshot* local e o mesmo se encarrega da atualização das réplicas quando seu tempo expira.

Já o problema apresentado na recuperação do *path* da imagem foi pesquisado a respeito e não teve como ser resolvido devido ao tempo disponibilizado pelas partes.

7 - CONCLUSÕES

O SGBD Oracle9i oferece ferramentas gráficas amigáveis com o usuário que facilitam a configuração do Oracle para operar como um SBDD. O SGBDD Oracle oferece níveis de transparência que proporcionam o sistema parecer para o usuário como um único SGBD local. Ele oferece também a replicação dos dados e possibilita a integração de banco de dados de outros fabricantes ao SBDD.

Com base nos testes apresentados pode-se concluir que o armazenamento da representação binária da imagem em *hosts* remotos é mais vantajoso que a utilização do armazenamento do *path* da imagem, pois o primeiro oferece tempos mais baixos e oferece consistência dos dados ao se armazenar a imagem no banco de dados. Já a utilização do armazenamento do *path* no *host* local apresentou uma performance significativa em comparação ao armazenamento local da representação binária da imagem, mas esta operação não leva em consideração a segurança dos dados. Portanto é aconselhável a utilização do tipo de dados `BLOB` para o armazenamento da imagem médica no banco de dados, devido o mesmo proporcionar segurança dos dados que é um aspecto que deve ser levado em consideração na implementação de sistemas PACS.

Para trabalhos futuros propõe-se a utilização da replicação avançada do Oracle, para que possa ser solucionada a restrição apresentada pelos tipos de dados *LOBs*. É interessante, na seqüência, fazer uma comparação entre os softwares utilizados nesse projeto e os novos softwares que deverão ser gerados para a inserção e recuperação dos dados nas réplicas da tabela mestra, e apresentação da conclusão sobre as comparações dos tempos obtidos. Também novos testes deverão ser realizados para descobrir novas formas de resolução dos problemas encontrados na recuperação da imagem utilizando-se o *path* remoto.

REFERÊNCIAS BIBLIOGRÁFICAS

CLUNIE, D. A. **DICOM Structured Reporting. PixelMed Publishing**. Disponível em: <http://www.dclunie.com/pixelmed/DICOMSR.book.zip>. Acesso em: 22 out. 2005.

CLUNIE, D. A. **MEDICAL IMAGE FORMAT FAQ**. 15 jun. 2004**.** Disponível em: <http://www.faqs.org/faqs/medical-image-faq/part1/>. Acesso em: 22 out. de 2005.

COELHO, Alex. **Sistema de auxílio à localização no CEULP/ULBRA utilizando imagens vetoriais scalable vector graphics**. Disponível em: <http://www.ulbra-to.br/ensino/43020/artigos/anais2004/anais/alexCoelhoSVGEncoinfo2004.pdf> . Acesso em: 05 nov. 2005.

CSEE. **Oracle Ducumentation**. Disponível em: <http://www.csee.umbc.edu/help/oracle8/index.htm>. Acesso em: 20 fev. 2005.

DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. Editora Campus, 2004.

DEITEL, H. M.; DEITEL, P. J. **Java: Como Programar (4ª Edição)**. Editora Bookman, 2003.

DEVAKI V; PARGAONKER, V; JAGANNATHAN, Y; AL-SALQAN, Tad Davis. **Intelligent Agents, HL7 and Patient Record Databases.** Abril 1995. Disponível em: <http://www.cerc.wvu.edu/cercdocs/techReports/1995/cerc-tr-rn-95-003.pdf>. Acesso em 02 de outubro de 2005.

ELMASRI, Ramez E.; NAVATHE Shamkant. **Sistemas de Banco de Dados**. Editora Addison Wesley, 2005.

GRAPHICS. **Graphics.** Jun. 1997. Disponível em: <http://www.dcs.ed.ac.uk/~mxr/gfx/>. Acesso em: 05 de nov. 2005.

HÓLM, Bjarki, et al. **Oracle 9i Java Programming**. Editora: wrox, 2001.

HORSTMANN, Cay S.; CORNELL, Gary. **Core Java 2 – Volume II – Recursos Avançados**. Editora Makron Books, 2000.

KELNER, J; FRERY, A.C.. **Realidade Virtual e Multimídia – Conceitos sobre imagens e o formato JPEG**. Disponível em: <http://www.cin.ufpe.br/~if124/mult_imagem.htm>. Acesso em: 05 de nov. 2005.

KOBAYASHI. L. O.; FURUIE, S. S.; GUTIERREZ, M. A. **Integração de modalidade utilizando um sistema de aquisição, Armazenamento e Transmissão de Imagens Médicas**. XVIII Congresso Brasileiro de Engenharia Biomédica. Paraíba – PB. vol. 15, p.271-275, 2002.

LEIDON, Universiteit . **Oracle9*i* Database Books**. Disponível em: <http://www.lc.leidenuniv.nl/awcourse/oracle/nav/docindex.htm>. Acesso em: 20 mar. 2005.

MARIANO, M. **Formatos de imagens: GIF e JPEG**. Disponível em: <http://www.mariomarino.com.br/digital/formatos.htm>. Acesso em: 05 nov. 2005.

MORENO, Ramon Alfredo. **Visualizador Contextual de Imagens Médicas.** 2005. Defesa de Doutorado (Sistemas Eletrônicos) - Universidade de São Paulo – Escola Politécnica. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3142/tde-10062005-174802/>. Acesso em: 10 out. 2005.

NETO, G. H.; OLIVEIRA, W; VALERI, F. V. **Processamento e Armazenamento de Imagens Médicas**. Disponível em: <http://www.comp.ufla.br/infocomp/artigos/v3.1/>. Acesso em: 15 out. 2005.

NUNES, F. L. S.; SCHIABEL, H.; ESCARPINATI, M. C.; BENATTI, R. **Comparisons of different contrast resolutions effects on a computer-aided detection system intended to clustered microcalcifications detection in dense breasts images**, Journal of Digital Imaging, vol. 14 (2), suppl. 1, p.217-219, 2001.

ORACLE. **Oracle9*i* Database Release 9.0.1 Documentation**. Disponível em: <http://www.oracle.com/technology/documentation/oracle9i_arch_901.html>. Acesso em: 10 abr. 2005.

ÖZSU, M. Tamer; VALDURIEZ, Patrick. **Princípios de Sistemas de Bancos de Dados Distribuídos**. Editora Campus, 2001.

ROBERT H. et al. **The HL7 Clinical Document Architecture**. Disponível em: <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=130066>. Acesso em: 25 ago. 2005.

SANTOS, M.; RUIZ, E. S. **Distribuição de Imagens Médicas com Uso de Tecnologia Web: Um Servidor e Cliente DICOM**. XVIII Congresso Brasileiro de Engenharia Biomédica. Paraíba – PB. vol. 15, p.253-257, 2002.

SILBERSCHATZ , Abraham; KORTH, Henry F.; SUDARSHA, S. **Sistema de Banco de Dados**. Editora Makron Books, 1999.

SUN. **Sun Microsystem**. Disponível em: <http://www.java.sun.com>. Acesso em jun. 2005.

Anexo A – Codigo fonte da Classe Conexao que estabelece a conexão com o Banco de dados.

```
import java.sql.*;
import oracle.jdbc.util.*;
import javax.swing.JOptionPane;

public class Conexao  extends Object{
       public static Connection getConexao(String host, String port,
String service_name, String user, String password){

       String url = "jdbc:oracle:thin:@"+host+":"+port+":"+service_name;

       Connection con = null ;

       try{
              //registra o driver no gerenciador de driver
              Class.forName("oracle.jdbc.driver.OracleDriver");

              //estabelece a conexão e retorna um objeto do tipo
Connection
              con = DriverManager.getConnection(url,user,password);

       }catch (SQLException sqlEx){
              con = null;
              System.out.println("Conexão não pode ser realizada,
verifique se o usuário e senha estão corretos...");
              JOptionPane.showMessageDialog(null,"Conexão não pode ser
realzada, \nverifique se o usuário e senha estão corretos.","ERRO NA
CONEXÃO...",JOptionPane.ERROR_MESSAGE);
              sqlEx.printStackTrace();

       }catch ( ClassNotFoundException classEx){
              con = null;
              JOptionPane.showMessageDialog(null,"ERRO AO CARREGAR O
DRIVER, Contate o fornecedor ","ERRO NA
CONEXÃO...",JOptionPane.ERROR_MESSAGE);
              System.out.println("Classe Não Encontrada");
              classEx.printStackTrace();
       }finally{
              return con;
       }
      }
}
```

Anexo B – Classe OperacoesTCC que contem todos os métodos que acessam o Banco de Dados.

```
import java.io.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import javax.swing.JOptionPane;

public class OperacoesTCC{
	private String localHost;
	private String porta;
	private String servico;
	private String user;
	private String senha;

	public OperacoesTCC(String host, String porta, String servico,
String user, String senha){
		localHost		= host;
		this.porta		= porta;
		this.servico		= servico;
		this.user		= user;
		this.senha		= senha;
	}

	// insere o path da imagem no banco de dados

	public int loadPath(String enderArq, int quantArm, int
idCorrente, String tabela , int idtabela){
		int num = 0;
		int id = 0;

		String host = "";

		try{
		Connection conn =
Conexao.getConexao(localHost,porta,servico,user,senha);
			conn.setAutoCommit(false);

			if (idtabela==0){
				for (id=idCorrente+1; id <=
quantArm+idCorrente; id++){
					String sql = "INSERT INTO "+tabela+"
(ID,URL) VALUES ("+id+",'"+enderArq+"')";

					OracleStatement stm =
(OracleStatement) conn.createStatement();
					stm.executeUpdate(sql);
					conn.commit();
					stm.close();
				}
			}else{
				switch(idtabela){
					case 1: host = "LAB0624";break;
					case 2: host = "LAB0623";break;
				}
```

```
                for (id=idCorrente+1; id <= quantArm+idCorrente; id++){
                        String sql = "INSERT INTO "+tabela+" (ID,URL)
VALUES ("+id+",'"+enderArq+"')";

                        OracleStatement stm = (OracleStatement)
conn.createStatement();
                        stm.executeUpdate(sql);
                        conn.commit();
                        stm.close();

                        ClientArm client= new ClientArm(host,enderArq);
                                client.runClient();
                        }
                }
                conn.close();
        }catch (SQLException ex){
                ex.printStackTrace();
                return -1;
        }
        num = id - 1;
        return num;
}

//busca o path das imagens no banco de dados

public int savePath(int idtabela, String tabela){
        String sql = "";
        int cont = 0;
        String host ="";

        switch(idtabela){
                case 0: host = localHost;
                case 1: host = "LAB0624";
                case 2: host = "LAB0623";
        }

        sql = "SELECT ID,URL FROM " + tabela ;
        try {
                Connection conn =
Conexao.getConexao(localHost,porta,servico,user,senha);
                conn.setAutoCommit(false);
                OracleStatement os = (OracleStatement)
conn.createStatement();
                OracleResultSet ors = (OracleResultSet)
os.executeQuery(sql);

                while(!ors.next()){
                        Client client = new
Client(host,ors.getString(2),ors.getInt(1));
                        client.runClient();
                        cont++;
                }
                os.close();
                ors.close();
                conn.close();
        }catch( SQLException ex){
                ex.printStackTrace();
                return -1;
        }
        return cont;
```

```
	}

	public int save( int idtabela , String tabela){
		int codigoRet = 0;
		String fileName = "";
		String sql = "";

		try{
			Connection conn =
Conexao.getConexao(localHost,porta,servico,user,senha);
			conn.setAutoCommit(false);

			sql = "select id,blob_imagem from "+ tabela ;

			if (idtabela==0){
				OracleStatement os = (OracleStatement)
conn.createStatement(  ); //utilizo o prepared devido minha instrução
sql ter parametros
				OracleResultSet ors = (OracleResultSet)
os.executeQuery(sql);

				if (!ors.next()){
					System.out.println("\nRegistro não
encontrado\n");

	JOptionPane.showMessageDialog(null,"Registros não
Encontrados","ERRO",JOptionPane.ERROR_MESSAGE);
					codigoRet = -1;
				}

				do {
					fileName =
"TCC/Fig"+ors.getInt("ID")+".Tiff";
					BLOB blob = ors.getBLOB(2);
					InputStream is =
blob.getBinaryStream();
					FileOutputStream fos = new
FileOutputStream( fileName );

					int size = blob.getBufferSize();
					byte buffer1[] = new byte[size];
					long wcount = 0;
					int length;

					while ((length = is.read(buffer1, 0,
size)) != -1) {
						wcount += length;
						fos.write(buffer1,0,length);
					}
					//System.out.println(wcount + " bytes
escritos no arquivo: "+fileName);

					codigoRet++;
					is.close();
					fos.close();
				}while(ors.next());

				ors.close();
				os.close();
				conn.commit();
```

```
	}else {
		String sql1 = "CREATE TABLE AUXILIAR AS (" + sql + ")";

		OracleStatement os =
(OracleStatement)conn.createStatement();
		os.executeUpdate(sql1);

		String sql2 = "SELECT ID,BLOB_IMAGEM FROM AUXILIAR";
		OracleResultSet ors = (OracleResultSet)
os.executeQuery(sql2);

		if (!ors.next()){

	System.out.println("\nRegistro não encontrado\n");

	JOptionPane.showMessageDialog(null,"Registro não
Encontrado","ERRO",JOptionPane.ERROR_MESSAGE);
			return -1;
		}

		do {
			fileName = "TCC/Fig"+ors.getInt("ID")+".Tiff";
			BLOB blob = ors.getBLOB(2);
			InputStream is = blob.getBinaryStream();
			FileOutputStream fos = new FileOutputStream(
fileName );

			int size = blob.getBufferSize();
			byte buffer1[] = new byte[size];
			long wcount = 0;
			int length;

			while ((length = is.read(buffer1, 0, size)) != -1)
			{
				wcount += length;

				fos.write(buffer1,0,length);
			}

//System.out.println(wcount + " bytes escritos no arquivo: "+fileName);

			codigoRet++;
			is.close();
			fos.close();
		}while(ors.next());

		String sql3 = "DROP TABLE AUXILIAR";
		os.executeUpdate(sql3);

		os.close();
		ors.close();
		conn.commit();
	}
	conn.close();
	}catch (Exception e){
		codigoRet=-1;
		e.printStackTrace();
	}finally{
		return codigoRet;
```

```
	}
}

//Armazena imagem no banco

	public int load( String enderArq, int quantArm, int idCorrente,
String tabela){
		int codigoRet = 0;
		long wcount = 0;
		String sql1="";
		int id =0;

		try{
			Connection conn =
Conexao.getConexao(localHost,porta,servico,user,senha);
			conn.setAutoCommit(false);

			for (id=idCorrente+1; id <= quantArm+idCorrente;
id++){

				FileInputStream is = new FileInputStream(
enderArq );

				if (tabela=="RAIOX" || tabela=="IMAGEM"){


					sql1 = "begin INSERT INTO "+ tabela
+" (ID,BLOB_IMAGEM) VALUES (? ,empty_blob()) " +
								" return BLOB_IMAGEM
into ?; " + " end;";

					OracleCallableStatement ocs1 =
(OracleCallableStatement) conn.prepareCall(sql1);
					ocs1.setInt(1,id);

	ocs1.registerOutParameter(2,OracleTypes.BLOB);
					ocs1.executeUpdate();

					BLOB blob = ocs1.getBLOB(2);

					OutputStream os =
blob.getBinaryOutputStream();
					int size = blob.getBufferSize();
					byte buffer1[] = new byte[size];
					int length;

					wcount = 0;

					while(( length =
is.read(buffer1,0,size)) != -1){
						wcount += length;
						os.write(buffer1,0,length);
					}

					//System.out.println("bytes escritos
no campo blob: "+wcount);

					os.close();
					ocs1.close();
```

```
                            }else {


                                    String sql2 = "CREATE TABLE AUXILIAR
AS (SELECT * FROM IMAGEM )";

                                    OracleStatement stm =
(OracleStatement)conn.createStatement();
                                    stm.executeUpdate(sql2);

                                    sql1 = "INSERT INTO "+ tabela +" (ID)
VALUES ( "+id+" ) ";

                                    OracleStatement ostm3 =
(OracleStatement) conn.createStatement();
                                    ostm3.executeUpdate(sql1);
                                    ostm3.close();


                                    String sql3 = "begin INSERT INTO
AUXILIAR (ID,BLOB_IMAGEM) VALUES (?,empty_blob()) " +
                                                    " return BLOB_IMAGEM
into ?; " + " end;";


                                    OracleCallableStatement ocs2 =
(OracleCallableStatement) conn.prepareCall(sql3);
                                    ocs2.setInt(1,id);

       ocs2.registerOutParameter(2,OracleTypes.BLOB);
                                    ocs2.executeUpdate();

                                    BLOB blob = ocs2.getBLOB(2);

                                    OutputStream os =
blob.getBinaryOutputStream();
                                    int size = blob.getBufferSize();
                                    byte buffer1[] = new byte[size];
                                    int length;

                                    wcount = 0;

                                    while(( length =
is.read(buffer1,0,size)) != -1){
                                           wcount += length;
                                           os.write(buffer1,0,length);
                                    }

                                    //System.out.println("bytes escritos
no campo blob: "+wcount);

                                    String sql4 = "UPDATE "+tabela+" SET
BLOB_IMAGEM = (SELECT BLOB_IMAGEM FROM AUXILIAR WHERE ID = "+id+" )
WHERE ID = "+id;
                                    OracleStatement ostm =
(OracleStatement) conn.createStatement();
                                    ostm.executeUpdate(sql4);
                                    os.close();
                                    sql4 = "DROP TABLE AUXILIAR";
                                    ostm.executeUpdate(sql4);
```

```
                                    ostm.close();
                                    ocs2.close();
                             }
                             is.close();
                             conn.commit();
                      }
                      //System.out.println(wcount + " bytes written to
BLOB field");
                      conn.close();
                      codigoRet = id;
               }catch (Exception e ) {
                      codigoRet = -1;
                      e.printStackTrace();
               }finally {
                      return codigoRet;
               }
       }


       public int delete( ){
               int codigoRet = 0;
               String tabela = "";

               try{
                      Connection conn =
Conexao.getConexao(localHost,porta,servico,user,senha);
                      conn.setAutoCommit(false);

                      // Procedure PL/SQL
                      for (int i = 0; i<3; i++){

                             switch(i){
                                    case 0: tabela = "ULTRASOM"; break;
                                    case 1: tabela = "RAIOX"; break;
                                    case 2: tabela = "TOMOGRAFIA"; break;
                             }

                             String sql1 = " delete from "+tabela;
                             OracleStatement stm = (OracleStatement)
conn.createStatement();
                             stm.executeUpdate(sql1);
                             conn.commit();
                          stm.close();
                      }

                       conn.close();
               }catch (Exception e ) {
                      codigoRet = -1;
                      e.printStackTrace();
               }finally {
                      return codigoRet;
               }
       }


}
```

Anexo C – Código fonte da Classe Client (Cliente).

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class Client {
	private ObjectOutputStream output;
	private ObjectInputStream input;
	private FileInputStream fis;
	private String chatServer;
	private Socket client;
	private String fileName;
	private int id;

	public Client(String host, String fileName, int id){

		chatServer = host;
		this.fileName = fileName;
		this.id = id;
	}


	public void runClient(){
		try{
			client = new Socket( InetAddress.getByName(
chatServer ),12000);

			output = new ObjectOutputStream(
client.getOutputStream());
			output.flush();
			input = new ObjectInputStream(
client.getInputStream() );

			sendData( ) ;

			processConnection();
			closeConnection();
		}catch(EOFException eofException){
			System.out.println("Server terminated connection");
		}catch(IOException ioException){
			ioException.printStackTrace();
		}
	}


	private void closeConnection() throws IOException{
		output.close();
		input.close();
		client.close();
	}
```

```
        private void processConnection() throws IOException{

                        try{

                                File file = new File(fileName);
                                String nomeArq = file.getName();
                                file = null;
                                nomeArq = nomeArq.replace(".",(id+"."));

                                FileOutputStream fos = new
FileOutputStream(nomeArq);

                                int length;

                                long wcount = 0;

                                while(( length = input.read()) != -1){
                                        wcount += length;
                                        fos.write(length);
                                }

                        }catch( Exception e){
                                System.out.println("\nNenhum Objeto
Recebido");
                        }
        }

        private void sendData( ){
                try{
                        output.writeObject(fileName);
                        output.flush();
                }catch( IOException ioException){
                        System.out.println("\nErro escrevendo no objeto");
                }
        }

}
```

Anexo D – Código Fonte da Classe Server (Servidor)

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.sql.rowset.serial.SerialJavaObject;

public class Server {
	private ObjectOutputStream output;
	private ObjectInputStream input;
	private ServerSocket server;
	private Socket connection;

	public void runServer(){
		try{
			server = new ServerSocket(5000,100);

			while(true){
				System.out.println("\nEsperando conexão\n
");
				connection = server.accept();

				output = new ObjectOutputStream(
connection.getOutputStream()); //enviando
				output.flush();  //esvazia o buffer de saída
				input = new ObjectInputStream(
connection.getInputStream() ); //recebendo

				processConnection();
				closeConnection();
			}
		}catch(EOFException eofException){
			System.out.println("Cliente Termino a conexão");
		}catch(IOException ioException){
			ioException.printStackTrace();
		}
	}

	private void processConnection() throws IOException{
		String fileName="";

			try{
				fileName = (String) input.readObject();

				System.out.println(fileName);
				FileInputStream fis = new
FileInputStream(fileName);

				int length;

				while ((length = fis.read())!=-1){
					output.write(length );
					output.flush();
				}
```

```
                        }catch( ClassNotFoundException
classNotFoundException){
                                System.out.println("\nNehum objeto
recebido");
                        }catch( IOException ioException){
                                        System.out.println("\nErro ao
escrever no objeto");
                        }catch( Exception e ){
                                e.printStackTrace();
                        }

        }

        private void closeConnection() throws IOException{
                output.close();
                input.close();
                connection.close();
        }

        public static void main(String args[]){
                Server inst = new Server();
                inst.runServer();
        }

}
```

Anexo E – Programa principal para recuperar o tipo BLOB.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class BuscarBlob {

       public static void main( String args[]){

              OperacoesTCC oper;
              int numtab = 0;
              String tabela = "";
              int valor = 0;
              String fileName = "";
              String input = "";

              input = JOptionPane.showInputDialog("Quantos registros
Buscar?");

              try{

                     valor = Integer.parseInt(input);
                     oper = new
OperacoesTCC("LAB0625","1521","IMGMED","SYSTEM","manager1");

                     if (valor > 50){
                         if (valor > 100)
                                numtab = 3;
                            else
                                numtab = 2;
                     }else{
                         numtab = 1;
                     }

                     for (int j=0; j<numtab; j++){

                            switch(j){
                                   case 0: tabela = "RAIOX"; break;
                                   case 1: tabela = "ULTRASOM"; break;
                                   case 2: tabela = "TOMOGRAFIA"; break;
                            }

                            long inicio = System.currentTimeMillis();

                            int result = oper.save(j,tabela); //executa
o método que recupera a imagem do tipo blob e armazena no HD para ser
visualizada

                            long fim = System.currentTimeMillis();

                            long t = fim - inicio; // subtrai os valores
do tempo para obter o tempo da operação
                            double tempo = t/1000; // obtem o tempo de
execução em segundos

                            System.out.println ("Buscando BLOB\nTabela =
"+tabela+"\nRegistros = "+valor+"\nTempo = "+tempo +"\n");
```

```
                              JOptionPane.showMessageDialog(null,"Tempo:
"+ tempo +" segundos.");

                              if (result == -1 ){

       JOptionPane.showMessageDialog(null,"ERRO ao Copiar para o
arquivo","ERRO",JOptionPane.ERROR_MESSAGE);
                                     System.out.println("ERRO ao Copiar
para o arquivo");

                              }else{
                                     System.out.println("Processo
Concluído...");
                                     System.out.println(fileName);

       JOptionPane.showMessageDialog(null,"Processo
Concluído...","Sucesso",JOptionPane.INFORMATION_MESSAGE);
                              }

                       }
              }catch (Exception ex){
                     ex.printStackTrace();
              }
       }
}
```

Anexo F - Programa principal para inserção do tipo BLOB.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*; // utilizar o objeto Date()
import java.io.*;

public class InserirBlob {
       public static void main( String args[]){
              String input = "";
              String tabela = "";
              String fileName = "";
              int valor = 0;
              int continua = 1;
              int numtab = 0;
              int reginicio = 0;
              int tamanho = 0;

       while (continua==1){
              input = JOptionPane.showInputDialog("Quantidade: ");
              fileName = JOptionPane.showInputDialog("Path");

              try{
                   valor = Integer.parseInt(input);
                      OperacoesTCC oper = new
OperacoesTCC("LAB0623","1521","CARLOS","SYSTEM","manager");

                       //pega o tempo em milisegundos inicial

                      if (valor > 1000){
                          if (valor > 2500)
                             numtab = 3;
                         else
                             numtab = 2;
                      }else{
                                   numtab = 1;
                      }

                      tabela = "TOMOGRAFIA"; // é colocada a tabela do
site local

                      tamanho = (int) (valor / numtab) + (valor %
numtab);

                      for (int j=0; j<numtab; j++){

                             long inicio = System.currentTimeMillis();


               //Insere valor na tabela indicada

              int result = oper.load(fileName,tamanho,reginicio,tabela);
```

```
              //pega o tempo final em milisegundos

              long fim = System.currentTimeMillis();

              long t = fim - inicio; // subtrai os valores do tempo para
obter o tempo da operação
              double tempo = t/1000; // obtem o tempo de execução em
segundos

              System.out.println ("Inserir BLOB\nTabela =
"+tabela+"\nRegistros = "+result+"\nTempo = "+tempo +"\n");
                     JOptionPane.showMessageDialog(null,"Tempo: "+ tempo
+" segundos.");
              if (result == -1 ){
                     JOptionPane.showMessageDialog(null,"ERRO ao
carregar o arquivo","ERRO",JOptionPane.ERROR_MESSAGE);
                     System.out.println("ERRO ao carregar o arquivo");
               }else {

       JOptionPane.showMessageDialog(null,"Processo
Concluido...","Sucesso",JOptionPane.INFORMATION_MESSAGE);
                     System.out.println("Processo Concluido...");
                     reginicio = result;
              }

              switch(j){
                     case 0: tabela = "ULTRASOM"; break;
                     case 1: tabela = "RAIOX"; break;
              }
        }
        continua =
Integer.parseInt(JOptionPane.showInputDialog("Continuar\n1 - SIM\n2 -
NÃO "));

        if (continua == 1)
              oper.delete();

        }catch( Exception ex){
            ex.printStackTrace();
        }
         }

      }
}
```

Anexo G - Programa principal para inserção do *path*.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*; // utilizar o objeto Date()
import java.io.*;

public class InserirPath {

       public static void main( String args[]){
              String input = "";
              String tabela = "";
              String fileName = "";
              int valor = 0;
              int continua = 1;
              int numtab = 0;
              int reginicio = 0;
              int tamanho = 0;

       while (continua==1){
              input = JOptionPane.showInputDialog("Quantidade: ");
              fileName = JOptionPane.showInputDialog("Path");

              try{
                   valor = Integer.parseInt(input);
                   OperacoesTCC oper = new
OperacoesTCC("LAB0623","1521","CARLOS","SYSTEM","manager");

                   if (valor > 1000){
                         if (valor > 2500)
                            numtab = 3;
                         else
                            numtab = 2;
                   }else{
                         numtab = 1;
                   }
                   //alterada de site para site
                   tabela = "TOMOGRAFIA";

                   tamanho = (int) (valor / numtab) + (valor % numtab);

                   for (int j=0; j<numtab; j++){
                      long inicio = System.currentTimeMillis();
                      int result =
oper.loadPath(fileName,tamanho,reginicio,tabela);
                      long fim = System.currentTimeMillis();
                      long t = fim - inicio; // subtrai os valores do
tempo para obter o tempo da operação
                         double tempo = t/1000; // obtem o tempo de
execução em segundos

                         System.out.println ("Inserindo Path\nTabela =
"+tabela+"\nRegistros = "+result+"\nTempo = "+tempo +"\n");
```

```
                        JOptionPane.showMessageDialog(null,"Tempo: "+
tempo +" segundos.");
                        if (result == -1 ){

                      JOptionPane.showMessageDialog(null,"ERRO ao
carregar o arquivo","ERRO",JOptionPane.ERROR_MESSAGE);
                      System.out.println("ERRO ao carregar o arquivo");
                        }else {
                      JOptionPane.showMessageDialog(null,"Processo
              Concluido...","Sucesso",JOptionPane.INFORMATION_MESSAGE);
                      System.out.println("Processo Concluido...");
                      reginicio = result;
                       }


                      switch(j){
                             case 0: tabela = "ULTRASOM"; break;
                             case 1: tabela = "RAIOX"; break;
                      }
                }
        continua =
Integer.parseInt(JOptionPane.showInputDialog("Continuar\n1 - SIM\n2 -
NÃO "));

        if (continua == 1)
              oper.delete();

        }catch( Exception ex){
            ex.printStackTrace();
        }
          }
      }
}
```

Anexo H - Programa principal para recuperação do path.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*; // utilizar o objeto Date()
import java.io.*;

public class BuscarPath {

       public static void main( String args[]){
              OperacoesTCC oper;
              int numtab = 0;
              String tabela = "";
              int valor = 0;
              String fileName = "";
              String input = "";
              input = JOptionPane.showInputDialog("Quantos registros
Buscar?");
              try{
                     valor = Integer.parseInt(input);
                     oper = new
OperacoesTCC("LAB0623","1521","CARLOS","SYSTEM","manager");
                     if (valor > 1000){
                         if (valor > 2500)
                            numtab = 3;
                         else
                            numtab = 2;
                     }else{
                            numtab = 1;
                     }
                     //alterada de site para site
                     tabela = "TOMOGRAFIA";
                     for (int j=0; j<numtab; j++){
                            long inicio = System.currentTimeMillis();
                            int result = oper.savePath(j,tabela);
                                          long fim =
System.currentTimeMillis();
                            long t = fim - inicio;
                            double tempo = t/1000;

                            System.out.println ("Recuperando pelo
Path\nTabela = "+tabela+"\nRegistros = "+valor+"\nTempo = "+tempo
+"\n");
                            JOptionPane.showMessageDialog(null,"Tempo:
"+ tempo +" segundos.");
                            if (result == -1 ){

JOptionPane.showMessageDialog(null,"ERRO ao Copiar para o
arquivo","ERRO",JOptionPane.ERROR_MESSAGE);
                                          System.out.println("ERRO ao
Copiar para o arquivo");
                            }else{
                                   System.out.println("Processo
Concluído...");
                                   System.out.println(fileName);
```

```
                                JOptionPane.showMessageDialog(null,
"Processo Concluído...","Sucesso",JOptionPane.INFORMATION_MESSAGE);
                                }

                                switch(j){
                                        case 0: tabela = "ULTRASOM"; break;
                                        case 1: tabela = "RAIOX"; break;
                                }
                        }
                }catch (Exception ex){
                        ex.printStackTrace();
                }
        }
}
```

Anexo I – Código fonte da Classe ClientArm(Cliente).

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class ClientArm {
	private ObjectOutputStream output;
	private ObjectInputStream input;
	private FileInputStream fis;
	private String chatServer;
	private Socket client;
	private String fileName;

	public ClientArm(String host, String fileName){

		chatServer = host;
		this.fileName = fileName;

	}

	public void runClient(){
		try{
			client = new Socket( InetAddress.getByName(
chatServer ),13000);

			output = new ObjectOutputStream(
client.getOutputStream());
			output.flush();
			input = new ObjectInputStream(
client.getInputStream() );

			sendData( ) ;

			closeConnection();
		}catch(EOFException eofException){
			System.out.println("Server terminated connection");
		}catch(IOException ioException){
			ioException.printStackTrace();
		}
	}

	private void closeConnection() throws IOException{
		output.close();
		input.close();
		client.close();
	}
```

```
        private void sendData( ){
                String aux = "";
                try{

                        output.writeObject(fileName);
                        output.flush();

                        FileInputStream fis = new
FileInputStream(fileName);

                        int length;

                        while ((length = fis.read())!=-1){
                                output.write(length );
                                output.flush();
                        }

                }catch( Exception Exception){
                        System.out.println("\nErro escrevendo no objeto");
                }
        }

}
```

Anexo J – Código fonte da Classe ServerArm(Servidor).

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ServerArm{
       private ObjectOutputStream output;
       private ObjectInputStream input;
       private ServerSocket server;
       private Socket connection;

       public void runServer(){
              String fileName="";

              try{
                     server = new ServerSocket(13000,100);

                     while(true){
                            System.out.println("\nEsperando conexão\n
");
                            connection = server.accept();

                            output = new ObjectOutputStream(
connection.getOutputStream());
                            output.flush();
                            input = new ObjectInputStream(
connection.getInputStream() );

                            try{
                                   fileName = (String)
input.readObject();

                                   FileOutputStream fos = new
FileOutputStream("TCC/"+fileName);

                                   int length = 0;
                                   long wcount = 0;

                                   while(( length = input.read()) != -
1){
                                          wcount += length;
                                          fos.write(length);
                                   }
                                   fos.close();
                            }catch( ClassNotFoundException
classNotFoundException){
                                   System.out.println("\nNehum objeto
recebido");

              }catch(IOException ioException){
                     ioException.printStackTrace();
              }
       }
```

```
        private void closeConnection() throws IOException{
                output.close();
                input.close();
                connection.close();
        }

        public static void main(String args[]){
                ServerArm inst = new ServerArm();
                inst.runServer();
        }

}
```

Anexo K – Telas dos erros encontrados na configuração do Nome de Serviço de Rede Local.

Anexo L – Imagem no Formato TIFF.