FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
MANTENEDORA DO CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VANESSA ALVES PEREIRA

AVALIAÇÃO DO COMPORTAMENTO DO SISTEMA GERENCIADOR DE BANCO DE DADOS ORACLE EM RELAÇÃO AO BANCO DE DADOS DISTRIBUÍDOS.

MARÍLIA
2006

VANESSA ALVES PEREIRA

AVALIAÇÃO DO COMPORTAMENTO DO SISTEMA GERENCIADOR DE BANCO DE DADOS ORACLE EM RELAÇÃO AO BANCO DE DADOS DISTRIBUÍDOS.

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.

Orientadora:
Profª Drª. Fátima de Lourdes dos Santos Nunes Marques.

MARÍLIA
2006

PEREIRA, Vanessa Alves
Avaliação do Comportamento do Sistema Gerenciador de Banco de Dados em Relação ao Banco de Dados Distribuídos / Vanessa Alves Pereira / Fátima L. S. Nunes Marques. Marília, SP: [s.n], 2006.

Apresentação de Monografia (Graduação em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha.

1. SGBD Oracle 9i. 2. Sistema Distribuído. 3. Banco de Dados.

CDD:

VANESSA ALVES PEREIRA

AVALIAÇÃO DO COMPORTAMENTO DO SISTEMA GERENCIADOR DE BANCO DE DADOS ORACLE EM RELAÇÃO AO BANCO DE DADOS DISTRIBUÍDOS.

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

CONCEITO FINAL: _____(_____)

ORIENTADOR: _____
Profª Dra. Fátima L. S. Nunes Marques

1º EXAMINADOR: _____
Prof. Edmundo Sergio Spoto

2º EXAMINADOR: _____
Profª Kalinka Regina L. Jaquie Castelo Branco

Marília, _____de _____de 2006.

“Existem derrotas, mas não existe o sofrimento. Um verdadeiro guerreiro sabe que ao perder uma batalha está melhorando sua arte de manejar a espada. Saberá lutar com mais habilidade no próximo combate”.

Paulo Coelho
Na Margem do Rio Piedra Eu Sentei e Chorei

PEREIRA, Vanessa Alves, Avaliação do Comportamento do Sistema Gerenciador de Banco de Dados em Relação ao Banco de Dados Distribuídos. 2006. 135 folhas.
Monografia (Bacharelado em Ciência da Computação) – Curso Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

RESUMO

Com o avanço de novas tecnologias e com a criação de ferramentas mais poderosas, muitas empresas migraram suas informações de um Banco de Dados Centralizado para um Banco de dados Distribuído, tendo em vista que os Bancos de Dados Distribuídos podem estar na mesma rede, ou até mesmo em países diferentes, compartilhando informações de uma maneira transparente para os usuários. Este trabalho tem por objetivo apresentar o SGBD Oracle9i como sistema de Banco de Dados Distribuído para o armazenamento e recuperação de volumes de dados variáveis, realizando uma comparação considerando tipos de dados, quantidade de tabelas e relacionamentos diversos. Para atingir o objetivo proposto foram necessárias a instalação e configuração do Oracle9i Distribuído e a criação de uma ferramenta para a execução dos testes de desempenho de armazenamento e recuperação desses diversos dados variando-se a quantidade de máquinas.

Palavras Chaves: Banco de Dados: Centralizados e Distribuídos, Sistemas de Banco de Dados Distribuídos(SBDD), SGBD Oracle9i, Sistema Gerenciador de Banco de Dados(SGBD), Armazenamento e Recuperação de dados.

PEREIRA, Vanessa Alves, Avaliação do Comportamento do Sistema Gerenciador de Banco de Dados em Relação ao Banco de Dados Distribuídos. 2006. 135 folhas.
Monografia (Bacharelado em Ciência da Computação) – Curso Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

ABSTRACT

With the advance of new technology and with the creation of more powerful tools many enterprises migrate their information from a Centralized Database to a Distributed Database one, considering that the Distributed Database can be in the same network, as well as they can be in different countries, sharing information in an easy way to the users. Therefore, his study aims at introducing the SGBD Oracle 9i as a Distributed database for the storage and recovering the amount of variable data, making a comparison considering different kinds of database, quantity of tables and direct relationship. To reach the proposed objective the installation and configuration of the Distributed Oracle 9i were need and also the creation of a new tool that executes some tests of performance in storage and recovering these different data varying the number of machines.

Keywords: Database: Centralized and Distributed, Distributed Database System (SBDD), SGBD Oracle 9i, Generation System of Database (SGBD), Storage and Recovering of data.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SBDD: Sistema de Banco de Dados Distribuído

BDD: Banco de Dados Distribuído

BD: Banco de Dados

I/O: Entrada e Saída

FK: Chave Estrangeira (Foreign Key)

PK: Chave Primária (Primary Key)

SQL: Linguagem de Consulta Estruturada (Structured Query language)

DDL: Linguagem de definição de dados (data definition language)

DML: Linguagem de manipulação de dados (data manipulation language)

PL/SQL: Linguagem Procedural da Oracle (Procedural Language/Structured Query Language)

TCP/IP: Protocolo de Controle de Transmissão (TCP) e Protocolo Internet (IP)

SUMÁRIO

INTRODUÇÃO

Com as constantes evoluções tecnológicas e o crescimento econômico, organizações necessitam manipular volume de dados cada vez maior, com segurança e sem interrupções. Para isso, essas organizações necessitam de servidores que possam disponibilizar esses dados armazenados de maneira rápida e eficiente. Com os avanços nos estudos de Banco de Dados (BD), muitas filosofias acabaram surgindo e muitas técnicas de armazenamento e recuperação de informações despontaram.

Segundo Silberschatz *et al.* (1999), todos os sistemas de bancos de dados eram centralizados, ou seja, existia um único servidor (máquina), responsável pelo armazenamento de todas as informações. A desvantagem na utilização desse sistema era que, se essa máquina, por algum motivo falhasse, todo o restante do sistema ficava sem obter qualquer informação. Com a evolução, surgiram os Bancos de Dados Paralelos, onde se pode notar um aumento na velocidade de processamento e aumento da taxa de entrada/saída, por meio do uso em paralelo de diversas CPUs e discos, o que acarretou em muitas operações sendo realizadas simultaneamente, ao contrário do processamento serial.

A mais nova filosofia em Banco de Dados que vem sendo estudada por um grande número de pesquisadores e ganhando, cada vez mais, espaço nas organizações é a que está sendo chamado de SBDD ou, Sistema de Banco de Dados Distribuídos. Em um SBDD, o banco de dados é armazenado em diversos computadores, comunicando-se através de redes de alta velocidade ou linhas telefônicas. Eles não compartilham memória principal ou discos e podem variar de tamanho e função.

Geralmente, em empresas de médio e pequeno porte, ainda são usados bancos de dados centralizados, onde o usuário faz uma solicitação e uma instância coleta a informação

no servidor, que é local. No entanto, em algumas empresas de grande porte, o BD centralizado está sendo substituído pelo distribuído, por poder ser acessado de qualquer parte (global).

Como a informação vem sendo tratada com grande relevância para o sucesso das empresas hoje existentes no mercado, é de fundamental importância que se utilizem as melhores tecnologias para proteger, tratar e guardar essas informações. Os Sistemas Gerenciadores de Banco de Dados (SGBDs) constituem uma das soluções mais rápidas de acesso e mais segura para o armazenamento dessas informações.

1.1. Objetivos

Este trabalho tem como objetivo avaliar o SGBD Oracle9i distribuído, que hoje é empregado ativamente no mercado, observando sua capacidade e desempenho perante o armazenamento e recuperação de dados em diversas tabelas com diferentes relacionamentos e também diferentes volumes de dados.

Para alcançar tal objetivo foi elaborado um programa desenvolvido em Java a fim de avaliar sua performance perante um BDD. A linguagem foi escolhida devido à grande aceitação que tem no mercado atual.

1.2. Disposição do trabalho

Esta monografia está organizada da seguinte forma:

O Capítulo 2 descreve as formas de processamento, conceitos de Banco de Dados e conceitos de Sistema Gerenciador de Banco de Dados, descrevendo também as vantagens e citando alguns exemplos.

No Capítulo 3 são apresentados conceitos de Banco de Dados Distribuídos e Sistema Gerenciador de Banco de Dados Distribuídos, descrevendo vantagens e desvantagens.

No Capítulo 4 é apresentado um histórico do Sistema Gerenciador de Banco de Dados Oracle juntamente com suas características.

No Capítulo 5 descrevem-se as ferramentas desenvolvidas para operar no sistema de Banco de Dados Distribuído, as configurações do SGBD Oracle9i para que o mesmo opere de forma distribuída e as definições dos testes com a finalidade de fornecer os dados para análise e comparação.

No 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.

2. BANCO DE DADOS

2.1. Considerações Iniciais

Segundo Date (1990), um sistema de banco de dados é um sistema cujo objetivo global é manter as informações e torná-las disponíveis quando solicitadas.

Um sistema de banco de dados envolve quatro componentes principais: dados, hardware, software e usuários. Em sistemas grandes, os dados geralmente são integrados e também compartilhados. O termo integrado refere-se à unificação de diversos arquivos de dados que são distintos, eliminando-se redundância entre os mesmos; já o termo compartilhado quer dizer que parcelas isoladas de dados podem ser compartilhadas por diversos usuários em um banco de dados, sendo que todos os usuários podem ter acesso à mesma parcela de dados. O hardware é composto pelos sistemas de armazenamento secundários, como os discos magnéticos, e pelas unidades de execução, como os processadores e memória principal, onde se localiza o banco de dados. Os softwares, conhecidos como Sistemas Gerenciadores de Banco de Dados, são os responsáveis por atender às solicitações dos usuários, prover segurança e gerenciamento das atividades. E, por fim, os usuários, pertencentes a três classes: os programadores de aplicações, que são responsáveis pela definição dos programas de aplicações que utilizam o banco de dados; os usuários finais, que interagem com o sistema por meio de um terminal on-line com o banco de dados e por último os administradores do banco de dados, que são responsáveis pelo gerenciamento do banco de dados (DATE, 1990).

O crescente interesse das empresas e do mercado mundial nas informações tem determinado o surgimento de muitos estudos nesta área e, com isso, o desenvolvimento de conceitos, técnicas e sistemas de banco de dados.

2.2. Vantagens da Utilização de Banco de Dados

Os sistemas de banco de dados trazem diversos benefícios e vantagens sobre sua utilização, entre os quais pode se destacar (DATE, 1990):

· A redução da redundância de dados representa a diminuição nos custos de armazenamento, pois se um mesmo dado pode ser encontrado em vários locais diferentes, é possível ocasionar futuramente a divergência entre as cópias dos dados armazenados.

· A inconsistência de dados é uma conseqüência da redundância e pode ser evitada com uma integração maior dos sistemas, impedindo que os dados sejam atualizados em apenas parte do sistema.

· Problemas de acesso aos dados ou recuperação de informações em sistemas não computadorizados são bastante insatisfatórios; entretanto, com os sistemas de bancos de dados tais problemas podem ser resolvidos facilmente com o uso das linguagens de consultas disponíveis no mercado.

· Controle de concorrência que impede que uma transação interfira na outra.

· A segurança em um sistema de banco de dados pode ser definida de uma forma para cada usuário, impondo restrições de acesso a dados de acordo com o perfil do usuário. Sem tais restrições, todas as informações no sistema poderiam estar comprometidas.

2.3. Abstração de Banco de Dados

A utilização de um SGBD permite e tem como grande objetivo, na visão de um usuário, omitir certos detalhes de como os dados são armazenados, mantidos e recuperados. Isto é possível tendo em vista a utilização, pelo sistema, de níveis de abstração no acesso aos dados. Segundo Silberschatz *et al.* (1999), os SGBDs empregam três níveis de abstração no acesso aos dados: nível físico, nível lógico e nível de visão.

- ***Nível Físico***: descreve a forma de armazenamento dos dados.

- ***Nível Lógico***: descreve quais dados estão armazenados e os relacionamentos entre estes dados. Este nível é amplamente utilizado pelos administradores de bancos de dados.

- ***Nível de Visão***: é através deste nível que se fornece ao usuário final a possibilidade de visualização das informações mantidas no banco de dados. Neste nível é possível limitar o acesso ao banco de dados, liberando a visualização de apenas uma parte dos dados aos usuários.

Os níveis estão sendo mostrados na Figura 2.1.

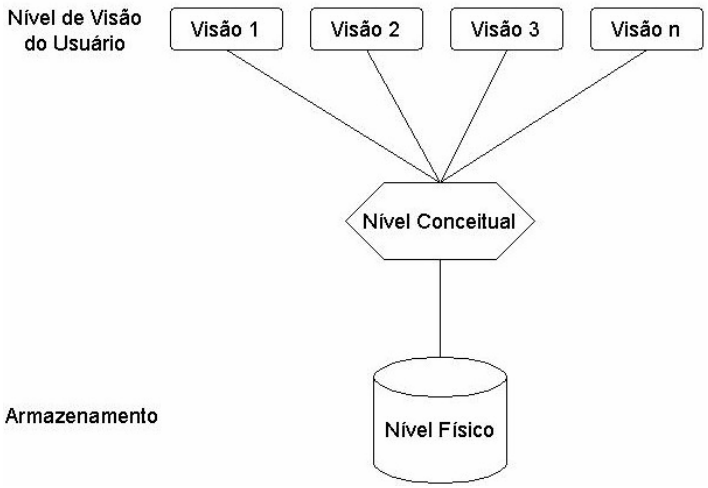


***Figura 2.1: Representação dos três níveis de abstração de dados (*Silberschatz *et al.,* 1999).**

2.4 Arquitetura Centralizada

Segundo Silberschatz *et al.* (1999), um sistema de banco de dados centralizado é aquele executado sobre um único sistema computacional, não interagindo com outros sistemas. Consiste basicamente de algumas CPUs e dispositivos de controle conectados através de um barramento comum de modo a proporcionar acesso à memória compartilhada. As CPUs possuem memórias cache locais no intuito de fornecer acesso

rápido aos dados através do armazenamento local de algumas informações. Os dispositivos de controle atendem a tipos específicos de dispositivos. Tendo em vista o fato de que a memória é centralizada, poderá existir alguma contenção no acesso concorrente à área de memória, acesso efetuado tanto por parte das CPUs quanto pelos dispositivos de controle. A memória cache exerce um papel fundamental na redução deste processo de contenção, pois permite que algumas informações possam ser alcançadas através da utilização da memória *cache*. Na Figura 2.2 é apresentado um exemplo de um sistema computacional centralizado.

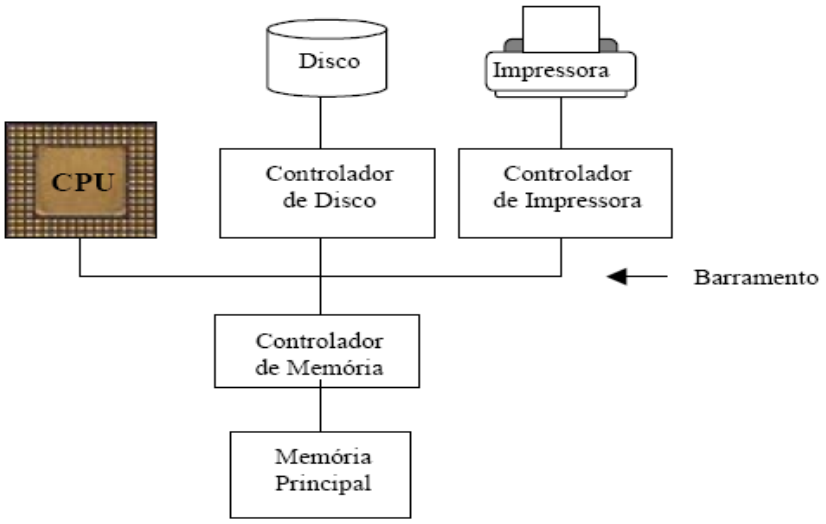


***Figura 2.2: Arquitetura de um sistema computacional centralizado (*Silberschatz *et al.*, 1999).**

Bancos de dados processados em equipamentos de um único processador, já dispõem de recursos multitarefas, permitindo que diversos processos sejam executados em um mesmo processador de modo compartilhado. A velocidade deste compartilhamento dá ao usuário a sensação de que estes processos estão sendo executados em paralelo. Desta forma, os equipamentos com paralelismo de granulação-grossa parecem idênticos a um equipamento de um único processador. Os equipamentos de granulação-fina possuem um grande número de

processadores. Os sistemas de bancos de dados rodando nesse tipo de equipamento podem processar consultas submetidas pelos usuários em paralelo (Silberschatz *et al.*, 1999).

2.5. Arquitetura Cliente/Servidor

Segundo Date (1990), sob um ponto-de-vista de mais alto nível, um sistema de banco de dados pode ser considerado como tendo uma estrutura muito simples dividindo-se em duas partes: um servidor e um ou mais clientes.

Os servidores, também conhecidos como *back-end*, são os responsáveis pelos gerenciamentos das requisições feitas pelos usuários, e os clientes, conhecidos como *front-end*, são responsáveis pela interação com o usuário. Na Figura 2.3 pode-se observar um exemplo da estrutura de um sistema cliente /servidor.

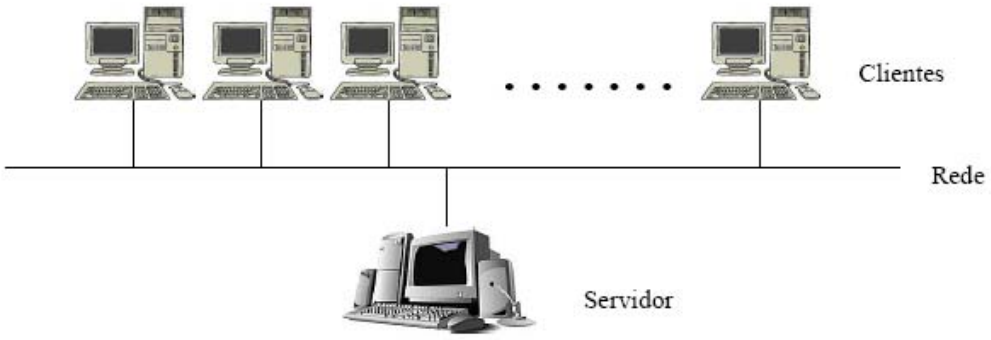


Figura 2.3: Estrutura de um sistema cliente/servidor centralizado(Date, 1990).

2.5.1. O Servidor

Um servidor de banco de dados tem como funções: estabelecer a integridade e segurança do banco, fazer o controle de concorrência e recuperação, manipular os dados

conforme as requisições dos clientes. Estão divididos em duas categorias: servidores de dados e servidores de transações.

Os servidores de dados são responsáveis pelas requisições de clientes que buscam ler, atualizar, remover ou inserir dados no banco. Necessitam de conexões velozes devido à grande quantidade de dados que podem ser solicitados por um usuário, já que todo o processamento é feito na estação cliente (SILBERSCHATZ *et al.*, 1999).

Os servidores de transações dão suporte às interfaces com os clientes, que são os meios pelos quais são feitos pedidos de determinadas ações para o posterior processamento pelo servidor. Ao término do processamento há o retorno do resultado da solicitação à estação que a requisitou (SILBERSCHATZ *et al.*, 1999).

2.5.2. O Cliente

Segundo Silberschatz *et al.* (1999), o sistema cliente é a parte responsável pela tarefa de requisição de pedidos ao servidor e também por toda a parte relativa à interação com o usuário final, ou seja, os clientes implementam ferramentas que ajudam visualizar o resultado de suas solicitações feitas ao servidor, como a geração de formulários, relatórios e interfaces do sistema.

Uma característica fundamental dos clientes é que toda a parte de comunicação entre os *back-ends* e os *front-ends* é transparente ao usuário. Significa dizer que o usuário tem a impressão de que tudo é feito na sua própria máquina.

2.6. Sistemas de Banco de Dados Distribuídos

Segundo Silberschatz *et al.* (1999), em sistemas de banco de dados distribuídos os dados podem ser armazenados em diversos computadores. A comunicação entre esses computadores se dá por troca de mensagens através de redes de alta velocidade ou linhas

telefônicas. Eles não compartilham memória principal ou discos. Na Figura 2.4 é ilustrada a arquitetura de uma rede com banco de dados distribuídos.

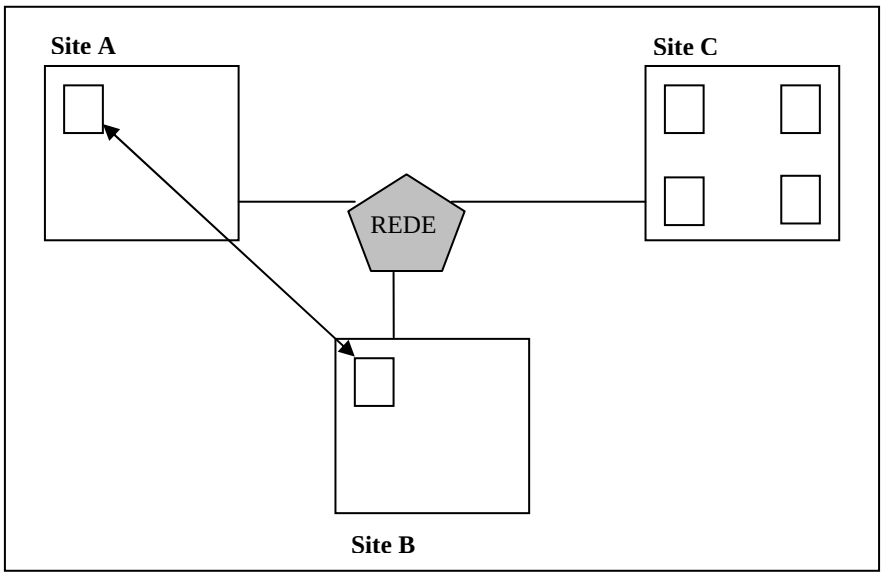


Figura 2.4: Estrutura de uma rede com Sistemas de Banco de Dados Distribuídos (Silberschatz et al., 1999).

Nos sistemas de banco de dados distribuídos encontram-se dois tipos de transações: transações locais e transações globais. As transações locais acessam um único computador e as globais acessam vários computadores (SILBERSCHATZ *et al.*, 1999).

2.7. Sistemas Paralelos

Sistemas paralelos caracterizam-se pela distribuição do processamento entre várias máquinas, permitindo uma maior velocidade ao processamento e às operações de entrada e saída (SILBERSCHATZ *et al.*, 1999).

A arquitetura paralela pode ser dividida ainda nas seguintes arquiteturas de bancos de dados paralelos: memória compartilhada, disco compartilhado, ausência de compartilhamento e hierárquico. Existem duas medidas principais para a avaliação do desempenho de um sistema de banco de dados: o *throughput*, que representa número de tarefas que podem ser

realizadas em um intervalo de tempo, e o tempo de resposta que mede o tempo gasto para o sistema processar uma única tarefa.

- ***Memória compartilhada*:** Segundo Silberschatz *et al.* (1999), na arquitetura paralela com memória compartilhada, os processadores e discos acessam a memória comum por meio de cabo ou alguma rede de interconexão. A vantagem da utilização da memória compartilhada é a eficiência na comunicação entre os processadores. Esta comunicação é feita através da troca de mensagens utilizando a memória. O grande problema deste tipo de arquitetura é no uso de mais de 32 ou 64 processadores, o *bus* (cabo) ou a interconexão de rede torna-se um gargalo do sistema, pois é compartilhado por todos os processadores. A partir de um determinado ponto, a inclusão de mais processadores não traz ganho para o sistema, visto que passarão a maior parte do tempo esperando por um momento de utilização do *bus* para o acesso à memória. Esse modelo é mostrado na Figura 2.5a.

- ***Disco compartilhado*:** neste modelo, segundo Silberschatz *et al.*, 1999, todos os processadores podem ter acesso direto a todos os discos, através de interconexão por rede, sendo que os processadores possuem memórias próprias. Visto que cada processador possui memória própria, nesta arquitetura não existirá o gargalo no *bus* para acesso à memória. Ainda neste modelo, a tolerância a falhas tem um ganho razoável, pois mesmo que ocorra uma falha no processador, ou na sua memória, outro processador poderá assumir estas tarefas e acessar o banco que continua residindo nos discos compartilhados. Existe ainda a possibilidade de através do RAID (*redundante arrays of inexpensive disks,* ou seja, técnica de conexão de diversos discos conectados a um sistema, proporcionando uma taxa maior de leitura e escrita dos dados, se operados em paralelo), oferecer uma tolerância a falhas para o subsistema de disco. Sistemas construídos sobre esta arquitetura são conhecidos como *clusters*. O grande problema desta arquitetura é na interconexão com o subsistema de discos, visto que a velocidade de comunicação em acesso a disco é infinitamente menor que no

tráfego de informações através da memória compartilhada. Esse modelo é mostrado na Figura 2.5b.

- ***Ausência de compartilhamento:*** na ausência de compartilhamento, segundo Silberschatz *et al.*, 1999, cada equipamento consiste em um processador, uma memória e discos. Os processadores dos nós podem se comunicar utilizando uma rede de alta velocidade. Cada nó tem a função de servidor dos dados mantidos nos seus discos. Desta forma, apenas as requisições para dados mantidos em outros nós irão sofrer com a queda de desempenho das operações de I/O executadas nos nós remotos. Esse modelo é mostrado na Figura 2.5c.

- ***Hierárquica*****:** esta arquitetura combina os compartilhamentos de memória e discos e o modelo sem compartilhamento. No nível mais alto, o sistema constitui-se de nós conectados por uma rede sem compartilhar discos ou memória entre eles. No topo da linha existe uma arquitetura sem compartilhamento. Cada nó do sistema pode ser um sistema com memória compartilhada entre alguns processadores ou, ainda, cada nó pode ser um subsistema de discos compartilhados e cada um desses sistemas que compartilham um conjunto de discos poderia também compartilhar memória (SILBERSCHATZ *et al.*, 1999). Esse modelo é mostrado na Figura 2.5d.

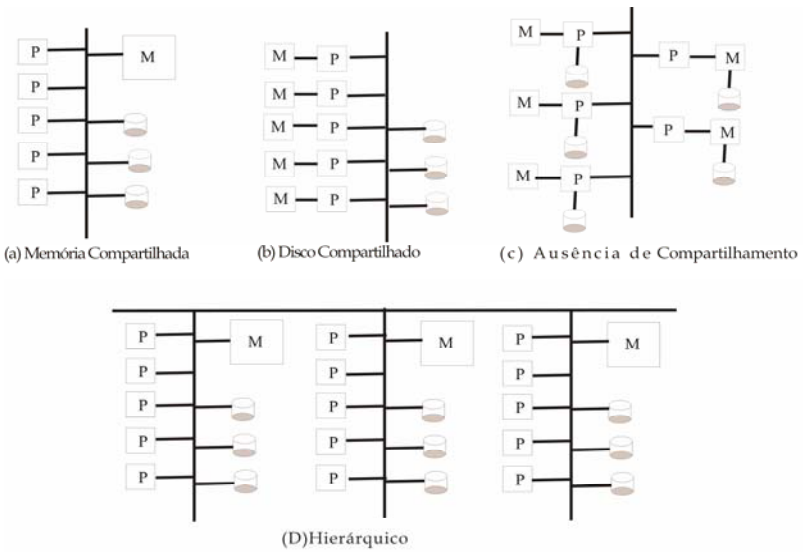


Figura 2.5: Arquitetura de banco de dados paralelos (Silberschatz et al., 1999).

2.8. Banco de dados relacional

Um banco de dados relacional, segundo Cerícola (1995), é uma coleção de dados organizados e integrados que são armazenados em forma de tabelas interligadas pelas chaves primárias e estrangeiras, constituindo uma representação natural dos dados, sem imposição de restrições ou modificações, a fim de que se possa adequar a qualquer computador para que possa ser usada por todas as aplicações, sem duplicação de dados e sem a necessidade de se definir programas, pois utiliza definições já existentes em sua base de dados, através do dicionário de dados ativo e dinâmico.

O método de acesso a dados relacional baseia-se na observação de que arquivos podem ser considerados como relações matemáticas; portanto, a teoria elementar de relações pode ser empregada para lidar com vários problemas práticos que surgem com esses dados. Assim, em tabelas relacionais suas linhas são chamadas de tuplas.

Na abordagem relacional todas as informações que estão em uma base de dados são representadas em forma de tabelas.

Para lidar com os dados que estão armazenados nessa tabela, os usuários contam com funções e operadores de alto nível. Quando se deseja efetuar mudanças nas definições dos bancos de dados, não há necessidade de haver mudanças nos programas de aplicação, pois pode-se inserir novas colunas nas tabelas, excluir colunas, excluir linhas, etc, que a aplicação continuará correta.

O objetivo principal do sistema relacional é fornecer um banco de dados de fácil compreensão e bem definido, facilitando, assim, sua manipulação.

2.9. Estrutura e Relacionamento

Em SGBDs relacionais, operadores permitem que qualquer tipo de acesso a um dado, ou a um conjunto desse seja possível (Álgebra Relacional). O dicionário de dados deve ser integrado, ativo e dinâmico para que os campos automaticamente fiquem consistentes.

As tabelas se relacionam umas com as outras através de chaves. Uma chave é um conjunto de um ou mais atributos que diferenciam os registros uns dos outros.

Existem dois tipos principais de chaves: a chave Primária (*PK - Primary Key*) que identifica cada registro como único e a chave estrangeira (*FK - Foreign Key*), que é um conjunto de atributos proveniente de outra tabela na qual está definido como chave primária. Define um relacionamento entre as tabelas, podendo ocorrer repetidas vezes.

Para otimizar as consultas são criados índices das tabelas, que consistem em estruturas de dados destinadas a facilitar a procurar de valores no conjunto de dados.

Hoje existem vários bancos de dados relacionais que são: DB2, InterBase, Oracle, Sybase, Firebird, SQL Server, PostgreSQL,etc (WIKIPÉDIA, 2006).

2.10. SQL (Structured Query language)

O nascimento da linguagem SQL, segundo Cerícola (1995), deveu-se à necessidade de executar as operações relacionais, incluindo Diferença, Divisão, Intersecção, Junção, Produto Cartesiano, Projeção e União. Essa linguagem suporta transações de processamento em caso de perda de dados. Pelo fato de ter sido adotada como a linguagem padrão para banco de dados, a SQL ocupa um lugar de destaque no processamento distribuído e na interligação entre computadores.

Em 1986 o primeiro padrão SQL1 foi adotado pela ANSI, tornando-se dominante no mercado. Esse padrão oficial foi considerado fraco e com limitada utilidade: por exemplo, não incluía comandos para alterar tabelas, comandos para excluir privilégios iterativamente, etc. Todos os problemas detectados na época além de outras funções foram adicionados gerando o SQL2. Novas "falhas" foram encontradas e por isso gerou-se a versão SQL3.

A linguagem SQL possui várias partes, cada qual composta por uma série de comandos específicos (NAVATHE E ELMASRI, 2000):

- **Linguagem de definição de dados (*data definition language* – DDL):** comandos para a definição de esquemas de relações, exclusão de relações, criação de índices e modificação nos esquemas de relações.

- **Linguagem de manipulação de dados (*data manipulation language* – DML):** comandos para inserção, exclusão e modificação de tuplas no banco de dados.

- **Definição de visões:** pode ser considerada uma parte da DDL, pois se preocupa da criação de visões no banco de dados.

- **Autorização:** comandos para a atribuição de direitos e permissões de acessos aos objetos do banco de dados.

- **Integridade:** comandos para a definição e imposição das restrições de integridade no banco de dados.

- **Controle de transações:** sob este item estão classificadas as estruturas de controle das transações, a concorrência e a atomicidade das transações são garantidas por este controle.

2.11. Estruturas Básicas para consulta

A estrutura básica de uma expressão de consulta em SQL consiste em três cláusulas:

***Select*:** corresponde à operação de **projeção** da álgebra relacional. Utilizada para identificar os atributos desejados para o resultado de uma consulta.

***From*:** corresponde à operação de **produto cartesiano** da álgebra relacional. Associa as relações que serão pesquisadas durante a execução da expressão.

***Where*:** identifica a seleção do predicado da álgebra relacional.

Considerando a seguinte consulta em SQL: ***Select*** A_1, A_2, A_3 ***From*** r_1, r_2, r_3 ***where*** P, a linguagem SQL construirá um produto cartesiano entre as relações constantes da cláusula ***From***, executa uma operação de seleção em álgebra relacional usando o predicado da cláusula ***where*** e, finalmente, projeta o resultado sobre os atributos da cláusula ***select***. A bem da verdade, existe um mecanismo de otimização do plano de execução da consulta, visando obter o plano de execução que ofereça o menor custo de processamento para a execução da consulta. Este plano de execução poderá alterar a ordem das operações que deverão ser efetuadas para a execução da expressão da SQL.

- **Cláusula Select**

O resultado de uma consulta é uma relação; uma relação pode ser considerada um conjunto, desta forma, dentro de uma relação não existirão tuplas duplicadas. A cláusula ***select*** indica o conjunto de atributos sobre os quais deve ser projetado o resultado de uma operação de seleção. Na cláusula ***select*** serão indicados os atributos para os quais a resposta da consulta deve ser projetada. Na cláusula, pode ser indicada uma série de atributos, podem ser indicados todos os atributos de uma relação através da utilização do símbolo *****. Ainda na cláusula ***select*** é possível envolver operações sobre o conjunto dos atributos. Desta forma, a

maioria dos SGBDs permite que a duplicidade de tuplas aconteça no resultado de uma cláusula ***select***, porém oferecem a possibilidade de eliminar a duplicidade através da palavra-chave ***distinct***. Na Figura 2.6 são apresentados alguns exemplos de comandos SQL.

1. **Select** cod_cliente, nome_cliente, fone_cliente **from** clientes
2. **Select distinct** cod_cliente, nome_cliente, fone_cliente **from** clientes
3. **Select * from** clientes
4. **Select** nome_cliente, saldo_cliente * 1.10 **from** clientes

Figura 2.6: Comando SQL Select e suas diversas utilizações.

- **Cláusula Where**

A cláusula ***where*** apresenta a seleção do predicado da expressão relacional e é através desta cláusula que o conjunto de resultados do produto cartesiano poderá ser restringido. Os operadores lógicos ***and, or*** e ***not*** poderão ser utilizados, bem como os operadores de comparação **=, <, >, <>, =>, <=.** Exemplos demonstrados na Figura 2.7.

1. **Select** cod_cliente, nome_cliente **from** clientes **where** cod_cliente >= 100
2. **Select** cod_cliente, nome_cliente **from** clientes **where** dt_nasc > ‘10/01/1980’ **and** cod_cliente < 1000.

Figura 2.7: Exemplo do uso da Cláusula Where.

- **Cláusula From**

A cláusula ***from*** define um produto cartesiano das relações na cláusula. Supondo que estivéssemos interessados em encontrar todos os nomes dos clientes e seus dependentes,

logicamente só dos clientes que possuem algum dependente. A consulta em SQL poderia ser expressa da seguinte forma (Figura 2.8):

```
Select clientes.nome_cliente, dependentes.nome_dependente from clientes, dependentes
where clientes.cod_cliente = dependentes.cod_cliente
```

Figura 2.8: Exemplo do uso da Cláusula From.

Na expressão da Figura 2.8 pode-se verificar a operação de produto cartesiano entre as duas relações **clientes** e **dependentes;** na cláusula ***where*** acontece a limitação do conjunto de resultados.

2.12. Ordenação de Tuplas

A cláusula ***order by*** permite que o resultado de uma consulta seja ordenado segundo algum atributo. Por padrão a cláusula ***order by*** classifica os resultados em ordem ascendente. Para que esta classificação mude para descendente será necessário incluir a palavra-chave ***desc*** na sintaxe da consulta. A ordenação pode ser realizada com base em um único atributo ou em uma série deles. A consulta que mostra o código do cliente e o seu nome, ordenados de forma descendente com base no nome do cliente, poderia ser expressa da seguinte forma (Figura 2.9):

```
Select cod_cliente, nome_cliente from clientes order by nome_cliente desc.
```

Figura 2.9: Exemplo do uso da Cláusula Order by.

2.13. Funções Agregadas

As funções agregadas possuem a capacidade de agregar, resumir um conjunto de valores relativos ao resultado de uma expressão, de forma a apresentar um único valor como resultado final. Como padrão, a SQL possui cinco funções agregadas (NAVATHE E ELMASRI, 2000):

- **Avg -** (average) – Média
- **Min -** (minimum) – Mínimo
- **Max -** (maximum) – Máximo
- **Sum -** (total) – Total
- **Count -** (count) – Contagem

A instrução ***Select avg(saldo_cliente) from clientes*** apresenta o valor da média dos saldos dos clientes, portanto, apresentará um valor único**.** Esta instrução apresenta como resultado um único atributo e uma única linha de registro. É possível a formação de grupos de tuplas, de forma que cada grupo apresente um valor resumido pela função de agregação. Isto é possível pela utilização da palavra-chave ***group by;*** o atributo ou atributos da cláusula ***group by*** irão indicar a formação dos grupos de resumos. Por exemplo, a relação **pedidos** identifica os pedidos emitidos para os clientes da empresa. Desta forma posso montar uma consulta que mostre o número de pedidos por código do cliente: ***Select cod_cliente, count(cod_pedido) from pedidos group by cod_cliente***. A função ***count*(*)** não elimina as duplicidades e não ignora os valores nulos.

A cláusula ***having*** atua nas funções de agregação da mesma forma que a cláusula ***where*** atua na restrição das tuplas. Seleciona os grupos após as suas formações, agindo desta forma, sobre os grupos e não sobre as tuplas que são envolvidas naqueles resumos de dados.

Por exemplo, podemos mostrar a quantidade de pedidos por código de cliente, porém somente daqueles clientes que possuem mais do que 5 pedidos (Figura 2.10):

```
Select cod_cliente, count(cod_pedido) from clientes group by cod_cliente
Having count(cod_pedido) > 5.
```

Figura 2.10: Exemplo do uso da Cláusula Having.

2.14. Visões de Dados

As visões trazem a possibilidade de esconder do usuário a complexidade do banco de dados, possibilitam a limitação do acesso aos dados. Na Linguagem SQL a criação de visões acontece através da seguinte sintaxe:

1. **Create view** nome_da_visão **as** < expressão da consulta >
2. **Create View** primeira_visao **as** Select * from clientes where cod_cliente > 10

Figura 2.11: Exemplo do uso de Visões.

É importante observar que as visões não armazenam dados, apenas contêm a "receita" de uma consulta a ser executada.

2.15. Manipulação de dados

Para incluir, excluir ou alterar dados em uma tabela usa-se comandos da DML. A operação de exclusão de dados pode ser expressa pela seguinte sintaxe (Figura 2.12):

Delete from nome da relação ***where*** predicado.

Figura 2.12: Sintaxe da Exclusão de Tuplas .

Por exemplo, para a exclusão de todos os registros de clientes cuja data de nascimento seja maior que 01/01/1980 a instrução será a seguinte (Figura 2.13):

Delete from clientes **where** dt_nascimento > ‘01/01/1980’

Figura 2.13: Exemplo de Exclusão de Tuplas .

A inserção de dados em uma relação tanto pode acontecer através da especificação das tuplas uma a uma, quanto pela inclusão do resultado de uma consulta. O comando de inserção deve fornecer todos os atributos necessários para a inclusão correta na relação e os valores a serem inseridos devem pertencer ao mesmo domínio dos atributos da relação onde se inserem os dados (Figura 2.14).

Insert into clientes (cod_cliente, nome_cliente, cpf_cliente) **values (** ‘1000’, ‘Jose da Silva’, ‘12345678911’) ;

Figura 2.14: Exemplo de Inclusão de Tuplas .

A operação de atualização permite que o usuário altere valores de alguns atributos da relação e (Figura 2.15).

Update nome_da_relação **Set** atributo = novo_valor **where** Predicado

Figura 2.15: Sintaxe de Atualização de Tuplas .

A seguir é apresentado um exemplo de atualização do valor do saldo dos clientes com código maior que 500 sendo que, o saldo sofrerá um acréscimo de 10 % (Figura 2.16).

```
Update clientes Set saldo = saldo * 1.10 Where cod_cliente > 500
```

Figura 2.16: Exemplo de Atualização de Tuplas .

2.16. Definição de Dados (DDL)

A Linguagem SQL contém uma série de instruções definidas para a construção do conjunto de relações e pode também definir o esquema das relações, domínio de cada atributo, regras de integridade, índices etc.

Para a definição dos domínios dos atributos das relações é necessário que se apresente uma relação dos principais domínios:

- ***Char(n):*** cadeia de caracteres de tamanho fixo de tamanho **n** a ser definido pelo usuário.
- ***Varchar(n):*** cadeia de caracteres de tamanho variável, com um tamanho máximo de **n** caracteres.
- ***Int:*** número inteiro válido
- ***Smallint:*** número inteiro, porém com uma abrangência menor que o tipo **int**
- ***Numeric(p,d):*** número de ponto fixo, o valor **p** identifica o número de dígitos, o valor **d** informa quantos dígitos estarão à direita do ponto decimal.

- ***Real, double precision:*** número de ponto flutuante com ponto flutuante de precisão dupla; esta precisão é dependente do equipamento.

- ***Float(n):*** número de ponto flutuante, a precisão **n** é definida pelo usuário.

- ***Date:*** calendário contendo um ano, mês e dia do ano.

- ***Time:*** *h*orário em horas, minutos e segundos.

Pode ser necessário proibir a permanência de valores nulos para determinados atributos. Para tanto a SQL permite que se especifique a condição ***not null*** no momento da criação da relação. Caso a lista de domínios não seja suficiente para os objetivos do usuário, é possível a criação de domínios "personalizados" através da instrução ***create domain*****.**

Create domain cpf **char(11)** -- esta instrução cria um domínio chamado **cpf** constituído de 11 caracteres.

a) Definição de Esquema

Uma relação (tabela) pode ser definida através do comando ***create table*** que possui a seguinte sintaxe (Figura 2.17):

Create table nome_da_relação **(** A_1 D_1, A_2 D_2, A_3 D_3, ..., A_n D_n, <regras de integridade$_1$>, <regras de integridade$_n$> **)**

Figura 2.17: Sintaxe de Definição de Esquema.

Na Figura 2.17, A identifica o nome do atributo e D o domínio daquele atributo. As regras de integridade podem ser a chave primária, as chaves estrangeiras, as restrições do tipo *check* etc. No momento iremos nos ater à restrição do tipo chave primária (*primary key*).

O comando ***primary key*** **(a1, a2, a3)** diz que a chave primária da relação será composta pelo conjunto de atributos a1, a2 e a3. Na Figura 2.18 é apresentado um exemplo da instrução de criação de relação.

```
Create Table funcionarios  ( cod_funcionario char(5) not null,
                             nome_funcionario varchar(30) not null,
                             fone_funcionario varchar(10),
                             primary key (cod_funcionario) )
```

Figura 2.18: Exemplo da Instrução de Criação de Relação.

A instrução ***drop table*** exclui a relação do banco de dados, elimina o esquema da relação banco de dados.

Neste capítulo foram apresentados os principais conceitos sobre Banco de Dados e Sistemas Gerenciadores de Banco de Dados. No próximo capítulo este assunto será aprofundado em relação aos conceitos de Bancos de Dados distribuídos.

3. BANCOS DE DADOS DISTRIBUÍDOS

3.1. Considerações Iniciais

Com as modernizações acontecendo a cada segundo é necessária uma agilidade no processamento de tarefas. Para tal finalidade é que equipes desenvolveram, e desenvolvem *softwares* e equipamentos para o acesso rápido e processamento paralelo de dados a qualquer distância.

Segundo Ozsu e Valduriez (2001), os sistemas distribuídos constituem conjunto de programas que executam simultaneamente em vários computadores interligados em rede, tentando fornecer ao usuário a impressão de que este é um sistema único e isolado. Outros dois conceitos importantes nesse sistema é o de Banco de Dados Distribuídos (BDD) e o de Sistema Gerenciador de Banco de Dados Distribuídos (SGBDD).

Pode-se definir um BDD como uma coleção de vários bancos de dados, logicamente inter-relacionados, distribuídos por uma rede de computadores. Já um SGBDD é um sistema que permite o gerenciamento do BDD e que torna a distribuição transparente para os usuários.

O BDD surgiu da integração da tecnologia de Banco de Dados Centralizado à de Rede de Computadores Distribuída, conforme mostrado na Figura 3.1.

Figura 3.1: Surgimento de Banco de Dados Distribuídos (Ozsu e Valduriez, 2001).

3.2. Estrutura de um SGBDD

Um SBDD consiste em uma coleção de nós, cada qual mantendo um Sistema de Banco de Dados local. Além de ser capaz de processar transações locais, cada nó ainda pode participar na execução de transações globais, ou seja, pode acessar dados localizados em outros nós. A execução de transações globais requer comunicações entre os nós através da rede (OZSU E VALDURIEZ, 2001).

Os nós no sistema podem ser conectados fisicamente de diversas formas, como mostrado na Figura 3.2. Uma borda do nó 2 para o nó 3 corresponde a uma conexão direta entre os dois nós. Algumas das configurações mais comuns são: redes totalmente conectadas, redes parcialmente conectadas, redes estruturadas em árvore, redes estrela e anel. A diferença entre essas configurações envolve itens como custos, disponibilidades e confiabilidades.

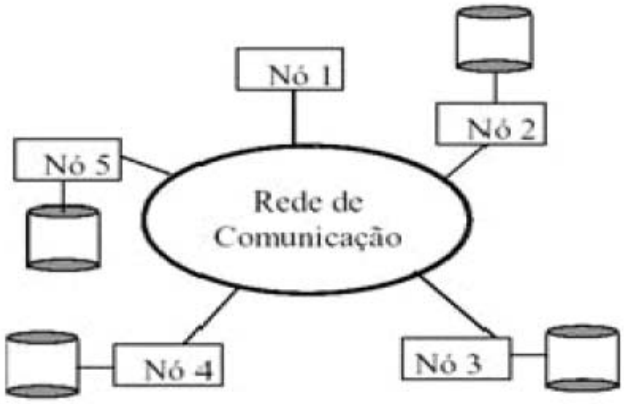


Figura 3.2: Caracterização de um ambiente em SBDD (Ozsu e Valduriez, 2001).

A principal vantagem de um SGBDD é partilhar e acessar dados de uma maneira confiável e eficiente, além do aumento de velocidade no processamento de consultas e do crescimento incremental, de possuir disponibilidade, confiabilidade e um controle distribuído,

além da sua transparência e autonomia. A seguir são detalhadas as principais características deste tipo particular de SGBD.

a) Compartilhamento de Dados e Controle de Distribuição

Com os dados compartilhados, um número de diferentes nós estão conectados; portanto um usuário em um nó deve poder acessar os dados disponíveis nos outros. Sendo assim, cada nó pode reter um grau de controle sobre os dados armazenados localmente. Em um sistema centralizado, o administrador de banco de dados controla todo o banco de dados. Em um sistema distribuído, existe um administrador de banco de dados global responsável pelo sistema inteiro, mas, partes destas responsabilidades são delegadas aos administradores locais de cada nó. Dependendo do projeto do sistema de banco de dados, cada administrador pode ter um diferente grau de autonomia local. A possibilidade de autonomia local é freqüentemente a maior vantagem de bancos de dados distribuídos (SILBERSCHATZ *et al.*, 1999).

b) Confiabilidade e Disponibilidade

Confiabilidade significa que um sistema funciona conforme foi projetado. Disponibilidade quer dizer que o sistema realiza suas funções sempre que é requerido. Ambas são necessárias, pois um sistema confiável que não esteja sempre disponível é inaceitável e um sistema disponível que não seja confiável também. Se um nó falha em um sistema distribuído, os demais nós devem ser capazes de continuar operando. Os dados estão replicados em vários nós; uma transação que necessita de um particular item de dado pode encontrá-lo em qualquer um dos outros nós. Assim, a falha de um nó necessariamente não implica na parada total de funcionamento de um sistema.

A falha de um nó deve ser detectada pelo sistema, e a apropriada ação deve ser recuperar o nó falho. O sistema não deve usar os serviços do nó falho enquanto este não esteja recuperado ou reparado. Mecanismos devem ser utilizados para reintegrá-lo ao sistema.

Embora a recuperação após falha seja mais complexa em sistemas distribuídos que em sistemas centralizados, a possibilidade de a maioria dos sistemas continuar operando apesar da falha de um nó resulta no aumento da disponibilidade, que, por exemplo, é crucial em sistemas de banco de dados utilizados em aplicações de tempo real. Outra vantagem seria o aumento da velocidade de processamento de consultas. Se uma consulta envolve dados de diversos nós, deve ser possível dividir a consulta em sub-consultas, ou seja, consultas menores, que podem ser executadas em paralelo.

c) Crescimento Incremental

Um sistema distribuído pode crescer mais facilmente que um sistema centralizado. Se for necessário expandir o sistema porque o volume de dados cresceu ou o volume de processamento aumentou, é mais fácil acrescentar um novo nó à rede de computadores, desde que os nó sejam autônomos, do que substituir um sistema centralizado já existente por outro maior.

d) Aceleração no Processo de Consultas

Se uma consulta envolve dados em diversos nós, é possível dividí-la em subconsultas que podem ser executadas em paralelo. Entretanto, em um sistema distribuído, não há o compartilhamento da memória principal, assim sendo, nem todas as estratégias para processadores paralelos podem ser aplicados diretamente a sistemas distribuídos (SILBERSCHATZ *et al*., 1999). Nesses casos em que os dados são duplicados, as consultas podem ser direcionadas pelo sistema para os nós com menos carga.

e) Transparência e Autonomia

A transparência de rede é o grau em que os usuários do sistema podem estar despreocupados com os detalhes do projeto do sistema distribuído, ou seja, é a separação entre a semântica de alto nível de um sistema e seus detalhes de implementação. A questão fundamental é prover independência de dados no ambiente distribuído (NAVATHE E ELMASRI, 2000).

Os usuários do banco de dados visualizariam uma única imagem da base de dados logicamente integrada, embora ela estivesse fisicamente distribuída. A autonomia local é o grau em que um projetista ou administrador de um nó pode ser independente do restante do sistema distribuído. Considerando as questões de transparência e autonomia, pode-se ter:

- **Autonomia de Nomeação e de Local**

Em bancos de dados distribuídos, cuidados precisam ser tomados para garantir que dois nós não usem o mesmo nome para itens de dados distintos. Uma solução para este problema é requerer que todos os nomes sejam registrados em um servidor de nome central (SILBERSCHATZ *et al.*, 1999). Para o aumento de autonomia local é requerer que cada nó prefixe seu próprio identificador para qualquer nome que ele gerar. Isto assegura que dois nós não gerem o mesmo nome (uma vez que cada nó tem um único identificador).

- **Transparência de Reprodução e Fragmentação**

Quando um item de dado é requisitado, a réplica específica não precisa ser nomeada. Em vez disso, uma tabela de catálogo é usada pelo sistema para determinar todas as réplicas para o item de dado. Da mesma forma, um usuário não deve ser solicitado a saber como um item é fragmentado (OZSU E VALDURIEZ, 2001).

- **Transparência de Localização**

A transparência de localização é obtida criando-se um conjunto de nomes alternativos ou *aliases* para cada usuário (SILBERSCHATZ *et al.*, 1999). Um usuário pode então se referir aos itens de dados por meio de nomes simples, os quais são traduzidos pelo sistema para nomes completos. Com *aliases*, o usuário pode despreocupar-se em relação à localização física de itens de dado. Ele não é afetado se o administrador do banco de dados decidir remover um item de um nó para outro.

- **Transparência e Atualização**

O problema principal é assegurar que todas as réplicas de um item de dado e todos os fragmentos afetados sejam atualizados. O problema de atualização para dados reproduzidos e fragmentados está relacionado ao problema de atualização de visões.

Todo sistema, apesar de eficiente, também têm suas desvantagens e algumas delas estão relacionadas aos custos e à complexibilidade, que pode gerar um aumento do baixo desempenho e de um grande potencial de "*bugs*" (NAVATHE E ELMASRI, 2000).

O custo em nível de *software* é bem mais alto em BDD do que em um banco de dados centralizado. Fatores que determinam isto são:

- O desenvolvimento de algoritmos eficientes e rápidos para controlar a fragmentação, replicação de dados e processamento de consultas;
- A prevenção de falhas em nós da rede. Quando houver falha, o sistema tem que continuar funcionando. Para isto, deve-se desenvolver um algoritmo contendo um número de replicação dos dados satisfatória para suprir esta necessidade;
- Os *softwares* desenvolvidos devem ser portáteis para serem executados em qualquer hardware e sistema operacional. O sistema de um ambiente para outro deve sofrer pequenas alterações;

- Os algoritmos de consulta devem ser rápidos e eficientes para satisfazer a expectativa de tempo de resposta que o usuário tem em relação ao sistema;
- Os algoritmos de *backup* devem ser capazes de guardar as últimas atualizações do sistema, além de especificar se os mesmos devem ser feitos à noite, ao meio-dia ou se devem parar todo sistema.

O mais importante é que o SGBDD e os programas devem garantir, acima de tudo, dados consistentes em todas as atualizações e consultas feitas.

A complexidade adicional requerida para assegurar a própria coordenação entre os nós é a principal desvantagem de um SBDD. Por meio dela são gerados um maior potencial de erros, uma vez que os nós que formam o sistema distribuído operam em paralelo, sendo mais difícil assegurar a correção dos algoritmos. Pode haver uma diminuição de desempenho de processamento uma vez que a mudança de mensagens e a computação adicional requeridas para realizar a coordenação interna não surgem em sistemas centralizados.

Dependendo de como a consulta foi desenvolvida (algoritmo) pode tornar-se muito complexa e levar muito tempo para ser executada (NAVATHE E ELMASRI, 2000).

3.3. Desenvolvimento de bancos de dados distribuídos

Os princípios de projeto de bancos de dados distribuídos incluem os mesmo princípios de projeto de bancos de dados centralizados, acrescidos de algumas questões específicas. Considerando-se uma relação *r* que será armazenada em um banco de dados, há diversas questões envolvidas em armazená-la em um banco de dados distribuído, incluindo as citadas a seguir.

3.3.1. Dicionário de Dados Distribuídos

A dificuldade para se implementar um banco de dados distribuído fica clara quando se começa a pensar na necessidade de conhecimento global sobre o sistema. Quando se olha do ponto de vista de comunicação, é difícil para um nó da rede saber "tudo" sobre os outros nós participantes. Detecção de *deadlock*, execução de transações distribuídas, são exemplos em que os nós da rede precisam se conhecer.

Estas questões não podem ser respondidas por um SGBD comum, que guarda informações sobre atributos, entidades, regras, índices, estatísticas, etc. Neste ponto, aplica-se o conceito de Dicionário de Dados Distribuído. Este é um dicionário de dados expandido, agora incluindo informações sobre dados remotos, informações de controle sobre os nós e a rede.

3.3.2. Replicação de Dados

O sistema mantém diversas réplicas idênticas (cópias) de uma relação. Cada réplica é armazenada em um nó diferente, resultando em replicação de dados. Se a relação *r* é replicada, uma cópia sua é armazenada em dois ou mais nós. Em casos mais extremos, há a replicação total, em que uma cópia é armazenada em todos os nós do sistema.

Há vantagens e desvantagens na replicação, como por exemplo:

Disponibilidade: se um dos nós contém a relação *r*, e esta falhar, *r* pode ser encontrada em outro nó. Assim, o sistema pode continuar a processar consultas envolvendo a relação r, apesar da falha de um nó.

Aumento do paralelismo: no caso em que a maioria do acesso à relação *r* resulta em uma leitura da relação, vários nós podem processar consultas envolvendo *r* paralelamente.

Quanto mais réplicas de *r* houver, maior é a chance de que os dados necessários sejam achados no nó onde a transação está executando. Portanto, a replicação de dados minimiza o movimento de dados entre os nós.

Aumento do *overhead* de atualização: o sistema deve assegurar que todas as réplicas de uma relação r estejam consistentes; caso contrário, processamentos errados podem acontecer. Assim, toda vez que *r* é atualizada, a atualização deve ser propagada a todos os nós que contêm réplicas. O resultado é um *overhead* aumentado.

Em geral, a replicação eleva o desempenho de operações de leitura e aumenta a disponibilidade de dados para transações. No entanto, transações de atualização incorrem em grande *overhead*. O problema de controlar atualizações concorrentes de várias transações para fazer a replicação de dados é mais complexo que o controle de concorrência em um banco de dados centralizado. Pode-se simplificar o gerenciamento das réplicas da relação *r*, escolhendo uma delas como a principal cópia de *r* (NAVATHE E ELMASRI, 2000).

3.3.3. Fragmentação de Dados

Uma relação *r* é dividida em diversos fragmentos. Cada fragmento é armazenado em um nó. Esses fragmentos contêm informações suficientes para reconstruir a relação *r* original. Esta reconstrução pode ocorrer pela aplicação da operação união ou um tipo especial de operação ligação sobre os vários fragmentos.

Na Tabela 3.1, mostra-se um exemplo de uma tabela de uma conta de um determinado banco, onde se tem os campos: Agência, Conta, Cliente e Saldo. Tendo esta tabela como exemplo, a Fragmentação pode ser dividida de três maneiras, descritas a seguir.

Tabela 3.1: Relação de exemplo conta.

Agência	Conta	Cliente	Saldo
Belém	0001	Dionísio	1000
Belém	0002	Josivaldo	2000
Manaus	0008	Paulo	800
Manaus	0009	Sandra	900

a) Fragmentação Horizontal

Divide a relação designando cada tupla de *r* para um ou mais fragmentos. A reconstrução de r pode ser obtida tomando a união de todos os fragmentos, como mostrado na Tabela 3.2.

Tabela 3.2: Fragmentação Horizontal da relação de conta.

Agência	Conta	Cliente	Saldo
Belém	0001	Dionísio	1000
Belém	0002	Josivaldo	2000

Agência	Conta	Cliente	Saldo
Manaus	0008	Paulo	800
Manaus	0009	Sandra	900

b) Fragmentação Vertical

Divide a relação decompondo o esquema R da relação *r*. A reconstrução de *r* pode ser obtida a partir dos fragmentos tomando a junção natural, como mostrada na Tabela 3.3 e na Figura 3.4.

Tabela 3.3: Relação de exemplo conta com identificação de tupla.

Agência	Conta	Cliente	Saldo	Tupla
Belém	0001	Dionísio	1000	1
Belém	0002	Josivaldo	2000	2
Manaus	0008	Paulo	800	3
Manaus	0009	Sandra	900	4

c) Fragmentação Mista

A relação *r* é dividida em uma série de relações fragmento. Cada fragmento é obtido como resultado da aplicação do esquema de fragmentação horizontal ou vertical na relação *r*, ou um fragmento de *r* obtido anteriormente.

Agência	Conta	Tupla
Belém	0001	1
Belém	0002	2
Manaus	0008	3
Manaus	0009	4

Junção dos fragmentos verticais

Conta	Saldo	Tupla
0001	1000	1
0002	2000	2
0008	800	3
0009	900	4

Figura 3.4: Fragmentação Vertical da relação conta.

3.4. Processamento de Consultas em Banco de Dados Distribuído.

Processar consultas em BDD corresponde à tradução de requerimentos formulados numa linguagem de alto nível sobre um nó da rede, numa seqüência de instruções elementares, as quais recuperam dados armazenados no BDD. Uma consulta pode ser também interpretada como qualquer acesso solicitado a algum repositório de dados (NAVATHE E ELMASRI, 2000).

O processamento de consultas em BDD é mais complexo que um BD centralizado. A eficiência de uma consulta em BDD depende de vários fatores combinados, citados a seguir.

- Se o BDD é heterogêneo ou homogêneo. Se for heterogêneo, haverá um custo de processamento adicional para que os banco de dados “conversem” entre si e troquem os dados necessários;
- Se há duplicação de dados e/ou os arquivos a serem consultados estão fragmentados e gravados pelos diversos nós da rede, isto implica que de alguma maneira os dados terão que ser agrupados para a resolução da consulta;
- A consulta deve ser otimizada, isto é, dividida em sub-consultas de maneira a diminuir o número de acessos ao disco e os produtos cartesianos entre as tabelas;
- A velocidade de acesso ao disco rígido, tempo de comunicação entre os nós, o número de produtos cartesianos e a maneira como a consulta foi elaborada são fatores influentes no tempo de resolução da consulta solicitada pelo usuário.

As consultas em um Banco de Dados Homogêneo são realizadas por um tradutor em cada nó que interpreta as instruções solicitadas, como por exemplo, duplicação de tabelas, comparações entre relacionamentos e operações lógicas (NAVATHE E ELMASRI, 2000).

Já as consultas em um Banco de Dados Heterogêneo, apesar de se ter também um tradutor em cada nó da rede, ele agora terá uma função muito mais crítica que a anterior. Ele tem que realizar a “conversação” entre os BDDs. O mapeamento de tabelas e a duplicação de

dados são tarefas complexas e o tradutor deve realizar esta tarefa de maneira rápida e eficiente (NAVATHE E ELMASRI, 2000).

A consulta em um BDD é realizada conforme o seu modelo relacional, contendo duas partes: especificar o domínio da relação a ser recuperada, ou seja, definir quais tabelas serão utilizadas e definir os predicados de seleção.

A complexidade da consulta aumenta com o aumento do número de operadores lógicos e operadores de comparação. O acesso aos nós da rede de BDD pode ser dividido em acesso centralizado e acesso distribuído (NAVATHE E ELMASRI, 2000).

A estratégia distribuída executa todas as sub-consultas nos diversos nós da rede juntamente com as operações lógicas e comparativas que puderem ser executadas em cada nó. A resposta final é gerada como a união das diferentes respostas de cada nó local da rede.

A estratégia centralizada consiste em agrupar para uma determinada consulta todas as sub-consultas e após, executar todas as operações lógicas em um simples nó. A característica principal deste tipo de consulta é que na fase final inclui-se um processamento único de consultas no local da consulta.

Em sistemas centralizados, o principal critério para medir o custo de uma consulta é o número de acessos a disco. Em sistemas distribuídos, acrescenta-se ao fato anterior, o custo de transmissão da rede e o desempenho de diversos nós executando processamento paralelo. Contudo, sabe-se que os três fatores citados dependem muito do hardware, software e tipo de redes instaladas.

Para se realizar uma consulta onde os dados estão fragmentados em diversos nós deve-se reunir todas as partes fragmentadas através de junções e uniões para reconstituir a tabela original. Se os dados estão duplicados, deve-se escolher a réplica mais atualizada possível para fazer a consulta. E, se os dados estão duplicados e fragmentados deve-se realizar as duas opções citadas (NAVATHE E ELMASRI, 2000).

3.5. Controle de Concorrência

O controle de concorrência é a parte de um SGBD capaz de assegurar que transações executadas em nós múltiplos produzam os mesmos resultados como se fossem executadas seqüencialmente num simples nó (SILBERSCHATZ *et al.*, 1999). O controle de concorrência possibilita o compartilhamento de dados de forma completamente transparente aos usuários, como pode ser visto em alguns conceitos, como:

- Integridade de Dados: assegurar que as informações que entram no sistema sejam verdadeiras e estão representadas concorrentemente. Um sistema bem desenvolvido deve permitir que se imponham restrições, que constituem regras que devem ser verificadas quanto aos dados;
- Consistência de Dados: mesma informação é guardada em diversos registros, possivelmente em arquivos diferentes. Seu valor deve ser o mesmo a qualquer momento em todos esses registros. Pode-se ter uma situação de inconsistência quando vários usuários estão utilizando concorrentemente o banco de dados, enquanto um usuário está alterando alguns registros, um ou mais usuários estão tendo acesso aos mesmos registros, sendo que alguns ainda não foram alterados.
- Objetos: menor unidade do banco de dados acessível pelo sistema de controle de concorrência (isto é, parte do BDD que se ocupa em decidir que ações devem ser tomadas em resposta aos requerimentos dos programas de aplicação para ler e gravar dentro do banco de dados). Os objetos podem ser arquivos, páginas, registros, etc.
- Ação: comando de processamento primitivo e indivisível, o qual é executado por um usuário simples.
- Operação: seqüência de ações que executam sobre um objeto uma função que respeita a consistência interna.

- Transação: é usada para descrever uma seqüência de operações sobre um ou mais objetos do banco de dados, transformando o atual estado consistente do sistema num novo estado consistente.
- Escalonador de Transações: controla o acesso de transações aos objetos do banco de dados. O escalonador é definido como responsável pela ordem na qual as diferentes operações para um conjunto de transações são feitas.
- Anomalias no Processamento Concorrente de Transações: produz-se como resultado da interferência entre os usuários que estão simultaneamente tendo acesso ao banco de dados. Essa interferência causa problemas de inconsistência, caracterizada pela violação das restrições de integridade. A finalidade principal do controle de concorrência é a prevenção dessa interferência. As anomalias pertencem a três tipos:
- (1) Perda de operações- ocorre na seqüência de eventos durante a execução de operações concorrentes; (2) Recuperações Inconsistentes- ocorrem devido à presença de restrições de integridade locais e/ou globais. Este problema aparece cada vez que uma transação tem acesso, ou pior ainda, modifica um estado transitório do banco de dados caracterizado pelo fato de que uma restrição de integridade não é verificada e (3) Conflito de operações- dá-se quando duas operações atuam sobre o mesmo dado e uma das operações é uma gravação. A ordem de execução das operações é computacionalmente significante se somente as operações estão em conflito.

Neste capítulo foram apresentados os principais conceitos e características sobre Banco de Dados Distribuídos. No próximo capítulo este assunto será aprofundado em relação ao Sistema Gerenciador de Banco de Dados Oracle.

4. SGBD ORACLE

4.1. Considerações Iniciais

O SGBD *Oracle* é um dos produtos mais utilizados no mundo todo para armazenamento e recuperação de dados. Sendo assim, não poderia deixar de ter, também, um dos mais complexos conjuntos de ferramentas e aplicativos para permitir a administração e manipulação dos seus dados. O *Oracle* fornece capacidade de interligação dos bancos de dados distribuídos como se estes fossem um só. Essas capacidades incluem pesquisas, inclusões, exclusões e transações distribuídas, retenção, replicação e particionamento em locais múltiplos.

Neste capítulo serão mostrados alguns conceitos do *Oracle* e como este SGBD trabalha com SBDD.

4.2. Histórico

Por volta de 1978, Larry Ellison visualizou uma oportunidade que outras companhias não haviam percebido quando encontrou uma descrição de um protótipo funcional de um banco de dados relacional e descobriu que nenhuma empresa tinha se empenhado em comercializar essa tecnologia. Ellison e os co-fundadores da Oracle, Bob Miner e Ed Oates, perceberam que havia um enorme potencial de negócios no modelo de banco de dados relacional, mas não se deram conta de que mudariam o conceito da computação empresarial para sempre (ORACLE CORPORATION, 2006).

A Oracle sempre foi uma empresa inovadora. Foi uma das primeiras empresas a tornar seus aplicativos empresariais disponíveis através da Internet - atualmente, esse conceito é predominante (ORACLE CORPORATION, 2006).

Recentemente a Oracle lançou o Oracle 10g Express Edition, uma versão gratuita do banco de dados da Oracle que oferece interfaces SQL e PL/SQL, além de ter suporte a linguagens como Java, .NET e PHP. O Database XE traz também o recurso Oracle HTML DB, que fornece um rápido desenvolvimento e implementação de aplicativos da web. O banco de dados está disponível para os sistemas operacionais Linux e Windows de 32 bits. A versão gratuita armazena, no máximo, 4GB de dados. Na Figura 4.1 é mostrada a interface do Oracle 10g XE (ORACLE CORPORATION, 2006).

Figura 4.1: Interface do Oracle Database XE.

4.3. Características do Oracle

Em um SBDD, funções que dependem do ambiente, tais como: *sysdate*, *user*, *uid*, etc, são sempre avaliadas no nó local.

Quando dados são requisitados de um banco de dados remoto, um servidor de banco de dados local se comunica com o banco de dados remoto usando uma rede de comunicação de *software* e o *ORACLE* SQL*Net.

SQL*Net é a interface usada para conectar clientes e servidores que operam sobre diferentes computadores da rede, e também conecta banco de dados servidores através de rede para facilitar as transações distribuídas.

O *Oracle* permite a mistura de diferentes versões, além de banco de dados de outras companhias para criar um Sistema Heterogêneo de SBDD (CERÍCOLA, 1995). Garante que cada banco de dados tenha um único conjunto de esquema e que dentro de cada esquema, um nome de objeto seja único dentro de seu lugar de nomes. Entretanto, cada esquema de nome de objeto é garantido para ser único dentro do contexto de qualquer banco de dados *Oracle* e o local pode facilmente resolver qualquer referência a objetos dentro de um banco de dados local.

Em um sistema de banco de dados distribuídos, cada banco de dados deve ter um único nome global. Nomes globais de banco de dados identificam cada banco de dados no sistema. Um nome de banco de dados global é composto de dois componentes: o nome do banco de dados com no máximo oito caracteres e o domínio da rede que contém o banco de dados. O componente domínio da rede de um nome global de um banco de dados deve seguir o padrão de convenção internacional de rede.

Cada dicionário de dados local num banco de dados distribuído ORACLE armazena somente nome de objetos e nomes de esquemas contendo nomes de objetos globais incompletos. Contudo, porque cada banco de dados pode ter um único nome dentro de uma rede e porque cada nome de objeto é garantido para ser único dentro do escopo de um simples banco de dados, cada objeto em um banco de dados distribuído ORACLE tem um único nome do objeto global.

Domínios de uma rede são similares aos diretórios de arquivos usados em muitos sistemas operacionais, tais como UNIX.

4.3.1. Tipos de ligações de banco de dados

O primeiro tipo é a ligação Privada que é criada em nome de um usuário específico. Uma ligação privada de banco de dados pode ser usada somente quando o proprietário da ligação especifica um nome de objeto global em uma instrução SQL (instrução GRANT ou visões, por exemplo).

O segundo tipo é a ligação Pública que é criada para um grupo especial de usuário público. Uma ligação Pública de banco de dados pode ser usada quando qualquer usuário no banco de dados associado especifica o nome do objeto global em uma instrução SQL ou definição de objeto.

E a terceira e última ligação de Rede que é criada ou gerenciada por um serviço de domínio de rede. Uma ligação de Rede de banco de dados pode ser usada quando qualquer usuário de qualquer banco de dados em rede especifica um nome de objeto global em instrução SQL ou definição de objeto.

4.4. Banco de Dados Distribuídos no Oracle

Na arquitetura cliente-servidor, o sistema de banco de dados Oracle é dividido em duas partes: um *front-end* com a parte do cliente e um *back-end* com a parte do servidor. A parte do cliente é a aplicação de banco de dados *front-end* que interage com o usuário. O cliente não tem qualquer responsabilidade de acesso aos dados e cuida raramente da solicitação, do processamento e da apresentação de dados gerenciados pelo servidor. Este, por sua vez, executa o Oracle e cuida das funções relacionadas com o acesso compartilhado concorrente (CERÍCOLA, 1995).

O Oracle utiliza um protocolo de *commit* de duas fases para lidar com transações concorrentes distribuídas.

Na arquitetura de banco de dados distribuídos Oracle, um nó pode atuar como cliente, como servidor, ou como ambos. Todos os banco de dados Oracle em um sistema distribuído utilizam o software de redes *Net8* para comunicação entre os bancos de dados. O *Net8* permite que bancos de dados se comuniquem através de redes para suportar transações remotas e distribuídas.

O Oracle suporta *links* de banco de dados que definem um caminho de comunicações de sentido único de um banco de dados do Oracle para outro.

Os dados podem ser replicados utilizando-se *snapshots* (instantâneos), ou tabelas-mestras replicadas.

Em banco de dados heterogêneos, ao menos um banco de dados é um sistema não-Oracle. O Oracle Open *Gateways* fornece acesso a um banco de dados não-Oracle a partir de um servidor Oracle que utiliza link de banco dados para acessar dados ou executar procedimentos remotos no sistema não-Oracle.

4.5. Estrutura de *hardware* e *software*

O Oracle contempla várias ferramentas para sua administração. Ferramentas como Server Manager, SQL*Plus e SQL*Net, que devem ser instaladas no servidor, são indispensáveis para a configuração do próprio banco e acesso aos dados nele contidos (CERÍCOLA, 1995).

Para acesso ao banco de dados, geralmente utiliza-se uma estação, como um PC comum, ligado ao servidor por um cabo de rede. Nessa estação deve estar instalado um *software* de sistema operacional ou ainda um *software* de emulação de terminal, para acesso direto com o servidor. Na estação também deve estar instalado o *software* responsável pela

comunicação com o banco de dados, o SQL*Net. É necessário, ainda, um *software* na estação para facilitar a pesquisa dos dados no banco.

4.6. Estrutura física e lógica

De acordo com Loney e Theriault (2002), fisicamente o SGBD ORACLE é composto de várias partes que são necessárias para o perfeito funcionamento das propostas do produto.

A primeira parte são os blocos de dados: um bloco de dados ORACLE é a menor unidade de I/O utilizado pelo banco de dados, também chamado de blocos lógicos e blocos ORACLE. Corresponde a um ou mais blocos físicos no disco. Seu tamanho é determinado na criação do banco de dados por um parâmetro de inicialização e é constante em todos os arquivos de dados. O seu tamanho também determina o tamanho de cada buffer de banco de dados na Área Global do Sistema (SGA, System Global Area).

A segunda parte são as extensões: uma extensão é um conjunto de blocos contíguos alocados para um segmento. Quando um objeto de banco de dados cresce, espaço é alocado para ele.

A terceira parte são os segmentos: um segmento é um conjunto de uma ou mais extensões que contém todos os dados para um tipo específico de estrutura de armazenamento lógico dentro de uma *tablespace*. Esses tipos são: segmento de dados; segmento de índices; segmento temporário; segmento de *rollback*.

A quarta parte é composta pelas *tablespaces*: um banco de dados ORACLE pode ser subdividido em áreas lógicas de espaço conhecidas como *tablespaces*. Cada *tablespace* contém um ou mais arquivos de sistema. Pode ser habilitada enquanto o banco de dados está ativo. Existem dois tipos de *tablespaces*: *tablespace SYSTEM* (que armazena dados que o

próprio sistema exige para ele mesmo, como o dicionário de dados) e tablespace não-*SYSTEM* (que armazenam os outros tipos de dados).

Logicamente, a forma de armazenamento dos dados em um SGBDR pode ser explicada por relações matemáticas, porém pode ser representada por conceitos mais claros e práticos.

Para a criação de um banco de dados dentro do SGBDR Oracle, primeiramente é necessária a definição de uma área para o armazenamento do dicionário de dados, uma área para os próprios dados e outra área para o controle de transações. Essas áreas são chamadas de "Espaço de Tabelas" (*Tablespaces*). Após essa definição, são criadas as "Tabelas" (*Tables*) dentro do espaço de tabelas correspondente, por exemplo, o de dados. Essas tabelas devem ser formadas de "Colunas" (*Columns*) e "Linhas" (*Rows*).

No momento em que o SGBD ORACLE começa a ser executado, a Área Global do Sistema é alocada na memória e os processos *background* são iniciados. A combinação do *buffer* de memória e os processos *background* é chamada de instância. A SGA é um grupo de *buffers* de memória compartilhada alocada pelo ORACLE para uma instância. Os processos *background* executam assincronamente tarefas distintas em benefício de todos os usuários do banco de dados.

Como todo banco de dados relacional, o SGBD Oracle trabalha por meio de consultas e atualizações de tabelas. Essas tabelas podem armazenar dados tanto de sistemas específicos, gerados por aplicações executadas por usuários, como dados das tabelas e outros controles do próprio SGBD. Quando uma tabela é criada, é necessária a especificação de cada tipo de dado da coluna. Os tipos de dados podem ser seguidos por um ou mais números entre parênteses que dão informações a respeito do tamanho da coluna. O tamanho da coluna determina o tamanho máximo de valores que ela pode ter.

As linhas de uma tabela são formadas pelos dados de cada coluna.

Com relação às versões, o Oracle possui várias opções como Universal Server, Express, Little, 7, 8i, 9i entre outras; e recentemente foi lançada a 10g. O “g” é de *Grid Computing* (Tipo de Sistema Paralelo e Distribuído) que tem como função “*Computational Grids*”.

De acordo com a Oracle Corporation (2006), as versões de comercialização do banco são *Standard e Enterprise*, sendo assim tem-se, por exemplo, o Oracle 9i *Standard e Enterprise*. A diferença entre um e outro está na quantidade de recursos (*Standard* é bem mais limitado) e quantidade de processadores que funciona (*Standard* funciona para até 4 processadores, *Enterprise* tem número ilimitado). Com relação às plataformas, o Oracle opera em Windows NT/2K/XP/2K3, Linux (executa em todas, caso sejam satisfeitos alguns requisitos, no entanto a Oracle só dá suporte às versões mais recentes do Oracle no RedHat Advanced Server 2.1 e united Linux) e UNIX (Solaris, HPUX, Tru64, etc..).

Com relação ao desenvolvimento de aplicações de bancos de dados distribuídos, ele possui recursos como Replicação, Streams, Database Link, e por possuir uma Máquina Virtual Java (JVM é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina) dentro de si, é possível trabalhar com RMI (Remote Method Invocation que é uma interface de programação que permite a execução de chamadas remotas no estilo RPC em aplicações desenvolvidas em Java. É uma forma de implementar um sistema distribuído em plataforma Java.).

Com relação aos protocolos de acesso estão disponíveis temos: TCP/IP, TCPS (TCP/IP c/ SSL), Named Pipes e IPC.

5. MATERIAIS E MÉTODOS

Este trabalho, como citado anteriormente, tem como objetivo avaliar o SGBD Oracle9i distribuído, observando sua capacidade e desempenho perante o armazenamento e recuperação de dados em diversas tabelas com diferentes relacionamentos e também diferentes volumes de dados.

Para atingir o objetivo almejado no presente trabalho foi feita a inserção, recuperação e exclusão dos dados em uma base de dados.

O ponto de partida foi a elaboração do modelo do banco de dados (Figura 5.1) e a definição de uma bateria de testes que pudessem fornecer subsídios para avaliar o comportamento do SGBDD Oracle em relação a aspectos diversos.

O modelo de banco de dados desenvolvido refere-se a uma concessionária de veículos. Para atender a necessidade da mesma, foram criadas doze tabelas.

Em uma concessionária os produtos são veículos e peças, que são vendidos separadamente; para isso criou-se a tabela de produtos com uma especialização, ou seja, tem-se uma tabela principal (*Produto)* que armazena as informações comuns a *Veículo* e *Peça* e em seguida criada essas duas tabelas, *Veículo* e *Peça*, para armazenar os dados específicos do produto.

Criou-se também a tabela *Montadora* onde foi preciso criar a tabela *Concessionária* para gravar as concessionárias que trabalham com tal montadora. Como cada montadora tem seus respectivos modelos de veículos, criou-se a tabela *Modelo*, onde estão armazenados os dados relevantes aos modelos dos veículos, que também está relacionada com a tabela *Veículo*. Sabe-se que cada modelo de veículo tem seus opcionais; por isso foi criada a tabela de *Opcionais* para registrar cada opcional em relação ao modelo do veículo.

E por último, para uma concessionária fazer suas vendas e registrá-las foi criado a tabela de *Vendas* juntamente com a tabela de *Itens da Venda*, e, como a venda está

relacionada a um cliente, criou-se a tabela *Cliente* contendo as informações relacionadas ao mesmo.

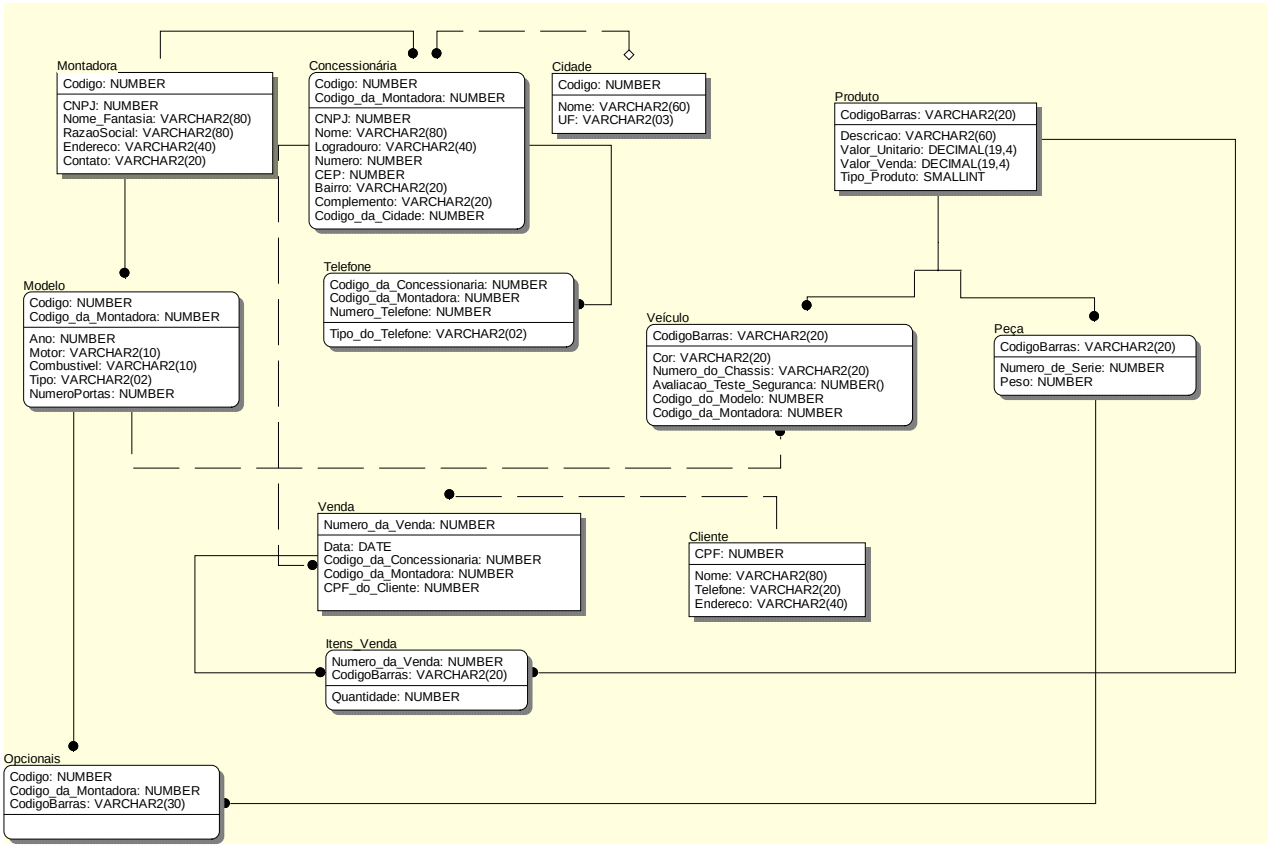


Figura 5.1: Modelo do Banco de Dados.

5.1. Instalação e Configuração do Oracle Distribuído

De acordo com Oracle Corporation (2006), a máquina deve ter as seguintes exigências mínimas:

- Sistema Operacional: Windows 98/NT/2000/XP.
- Sistemas de Arquivos: Oracle recomenda utilizar FAT 32 para Windows 98, e para demais utilizar NTFS.

- Hardware: 128 MB de RAM, recomendado 256 MB, 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 3.85 GB.
- Protocolos: TCP/IP, TCP/IP com SSL e Named Pipes.

Após ter definido o modelo do banco de dados foi realizada a instalação do SGBDD Oracle em cinco máquinas. Todas as máquinas utilizadas para os testes têm duas partições no disco rígido, onde em uma partição tem o Sistema Operacional Windows XP e na outra o Linux. As configurações dessas máquinas são:

- **Máquina lab0621**: Pentium 4, 2.80 GHZ, HD de 80 GB, e 512 MB de RAM, a marca do HD é Western Digital.
- **Máquina lab0622**: Pentium 4, 2.80 GHZ, HD de 80 GB, e 512 MB de RAM, a marca do HD é Western Digital.
- **Máquina lab0623**: Pentium 4, 2.80 GHZ, HD de 80 GB, e 512 MB de RAM, a marca do HD é Samsung.
- **Máquina lab0624**: Pentium 4, 2.80 GHZ, HD de 80 GB, e 512 MB de RAM, a marca do HD é Western Digital.
- **Máquina lab0625**: Pentium 4, 2.80 GHZ, HD de 80 GB, e 512 MB de RAM, a marca do HD é Western Digital.

As máquinas foram nomeadas como LAB0621, LAB0622, LAB0623, LAB0624 e LAB0625. O processo de instalação e configuração do SGBDD é detalhado a seguir.

Na Figura 5.2 é mostrado que existem três opções no início da instalação: iniciar a instalação, explorar os arquivos do CD de instalação ou informações de paginação (que se refere a toda documentação do Oracle9i).

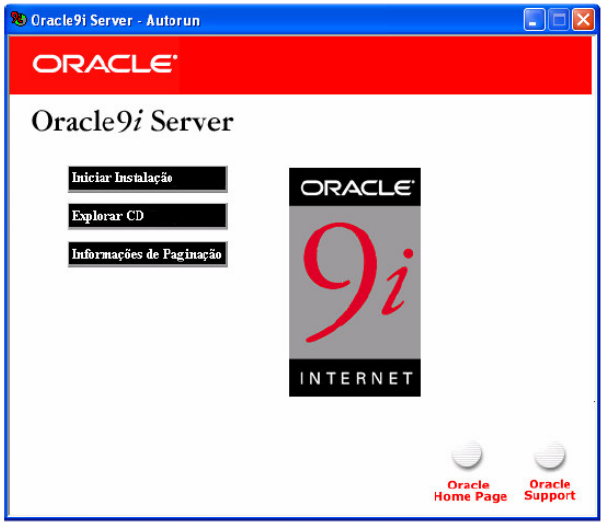


Figura 5.2: Tela inicial de instalação do Oracle9i.

Escolhendo a opção iniciar instalação, uma nova tela irá surgir, exatamente como mostrada na Figura 5.3.

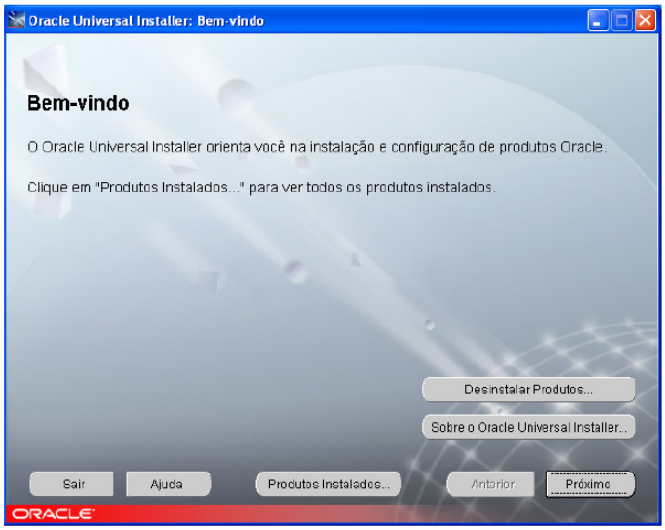


Figura 5.3: Tela de boas vindas do Oracle9i.

Na Figura 5.3 é exibida a tela de boas vindas do Oracle9i, por meio da qual é possível escolher entre eliminar produtos já instalados, visualizar produtos já instalados, abandonar a instalação e dar seqüência à instalação.

Ao escolher a opção *próximo* a tela apresentada será semelhante à Figura 5.4, onde deverá ser informado o local de onde os arquivos serão instalados

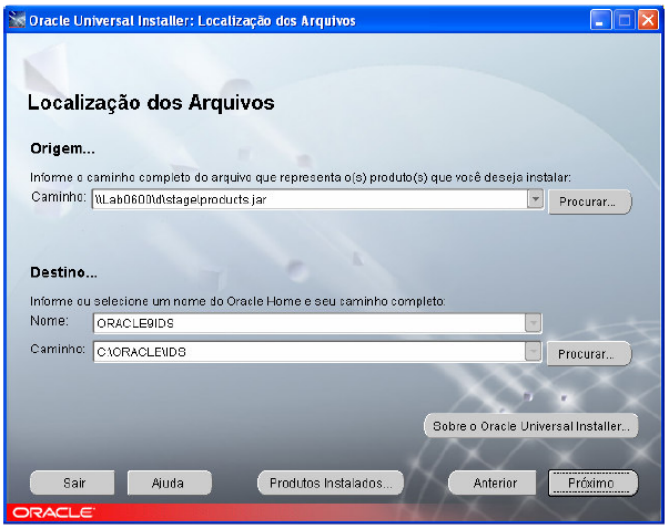


Figura 5.4: Tela de localização dos arquivos de instalação do Oracle9i.

Na Figura 5.5 é mostrada a próxima tela da instalação onde se pode escolher entre três tipos de produto para se instalar. A opção escolhida para desenvolver esse trabalho foi a primeira opção que inclui a instalação dos seguintes itens: um banco de dados pré-configurado; os serviços de rede; as ferramentas do ambiente de banco de dados; a estrutura do *Oracle Enterprise Manager* das ferramentas de gerenciamento, incluindo o Console, o *Management Server* e o *Intelligent Agent*; os utilitários Oracle e a documentação on-line.

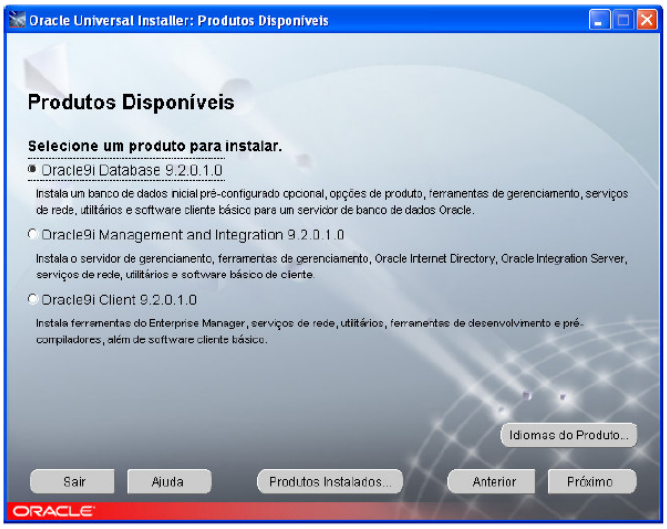


Figura 5.5: Tela de opções de produtos Oracle9i.

Na Figura 6.6 são mostrados os tipos de instalações disponíveis que o usuário pode escolher. A primeira opção seria a *Enterprise Edition* que 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 Utilites*, e a documentação on-line.

Na segunda 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, porém suportando apenas um único usuário que tenha compatibilidade total com as outras duas opções.

E finalmente a 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.

Para esse trabalho a opção escolhida foi a primeira opção (*Enterprise Edition*).

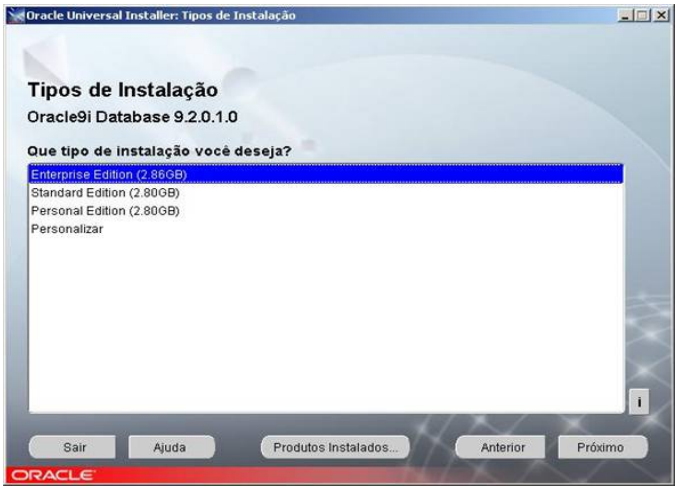


Figura 5.6: Tela de tipos de instalação do Oracle9i.

A próxima tela exibida no processo de instalação está reproduzida na Figura 5.7, onde o usuário pode escolher as configurações do banco que está sendo criado correspondendo às suas necessidades. Se o usuário optar pela opção *Finalidade Geral*, o banco de dados criado será otimizado para uso nas diferentes tarefas. Escolhendo a opção *Personalizada* o usuário decidirá como deve ser instalado o banco de dados e sua forma de otimizar o mesmo. Escolhendo a última opção (*Somente Software*), o usuário precisar instalar um banco de dados posteriormente, podendo usar a ferramenta *Database Configuration Assistent.*

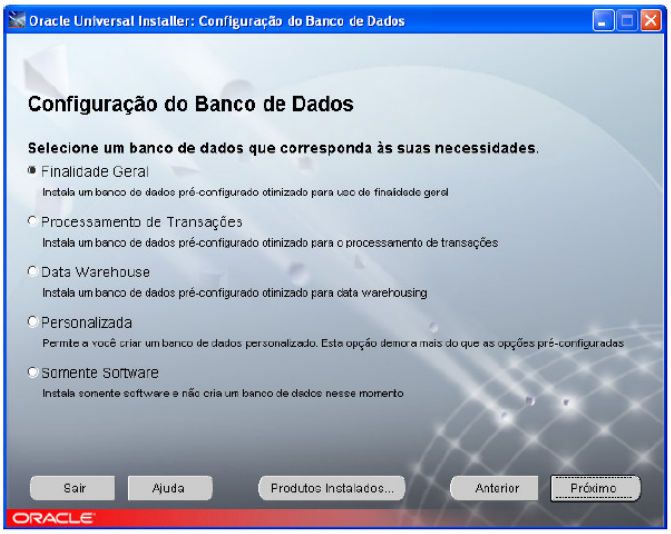


Figura 5.7: Tela de tipos de configurações disponíveis pelo Oracle9i.

Na próxima etapa (Figura 5.8) deverá ser informada a porta à qual o *Listener* irá usar para esperar por conexões entrantes. O *Listener* (ouvinte) é um processo que espera por conexões do cliente para acessar uma base de dados ou diversas bases de dados. Nesse trabalho foi escolhida a porta 1521 (default).

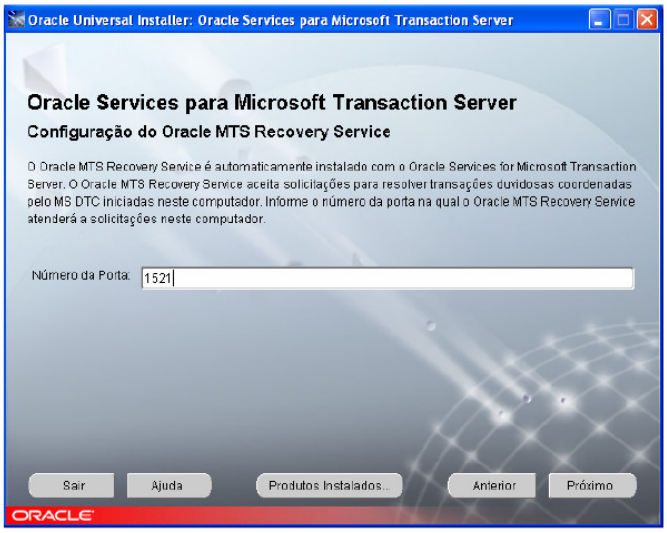


Figura 5.8: Tela da escolha da porta do Listener.

Na próxima fase é preciso informar o nome do banco de dados e o SID. Um SID (System Identifier Database) é um identificador exclusivo do banco de dados e não pode ser informado um identificador que já esteja em uso pelo Oracle (ORACLE, 2005). Esse processo pode ser visto na Figura 5.9.

O nome do banco de dados global informado para a máquina LAB0621 foi TCCBD21 e, ao colocar um nome do Banco de Dados global, um SID foi informado onde nesse trabalho ele foi aceito, mas no caso de não ser aceito pode ser colocado outro SID. Se essa descrição estiver sendo usada por outro banco de dados que já esteja instalado, o Oracle retornará um erro. Para a máquina LAB0622 foi informado o nome TCCBD22 e o SID TCCBD22; para a máquina LAB0623 foi informado o nome TCCBD233 e o SID

TCCBD233; para a máquina LAB0624 foi informado o nome TCCBD24 e o SID TCCBD24 e finalmente à máquina LAB0625 foi informado o nome TCCBD255 e o SID TCCBD255.

Figura 5.9: Tela de identificação do banco de dados.

Na etapa seguinte um local para a instalação dos arquivos do banco de dados deve ser informado. Nesse trabalho foi informado o caminho: C:\ORACLE\ORADATA, como mostrada na Figura 5.10.

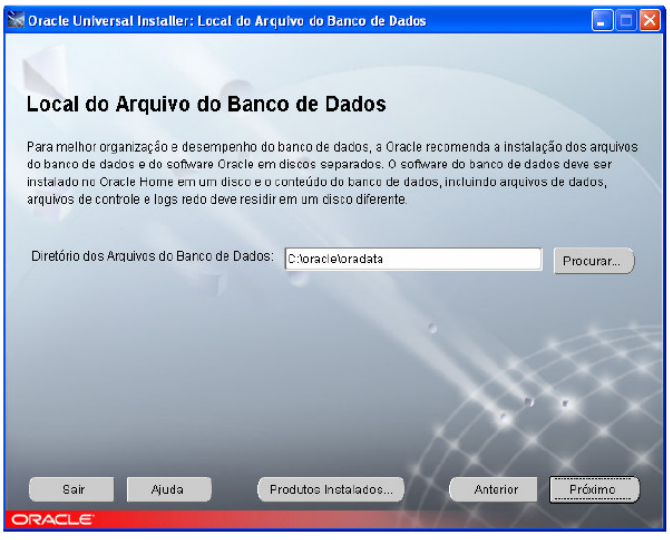


Figura 5.10: Tela para informar o diretório dos arquivos do banco de dados.

A próxima tela é para escolher o conjunto de caracteres que o banco de dados utilizará. A opção escolhida nessa etapa foi o conjunto *Default*, exatamente como mostrada na Figura 5.11.

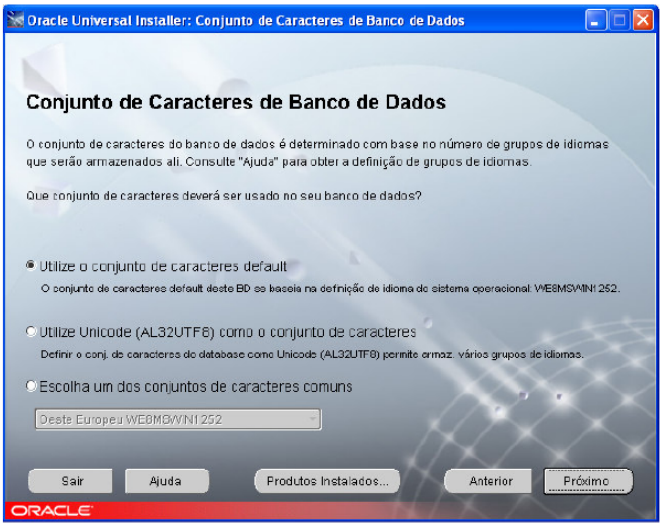


Figura 5.11: Tela para a escolha do conjunto de caracteres do banco de dados.

Seguindo a instalação, é mostrada uma tela com um resumo dos *softwares* e opções escolhidas que será instalado na máquina, como mostrada na Figura 5.12. Prosseguindo com a instalação serão instaladas as opções escolhidas anteriormente pelo usuário.

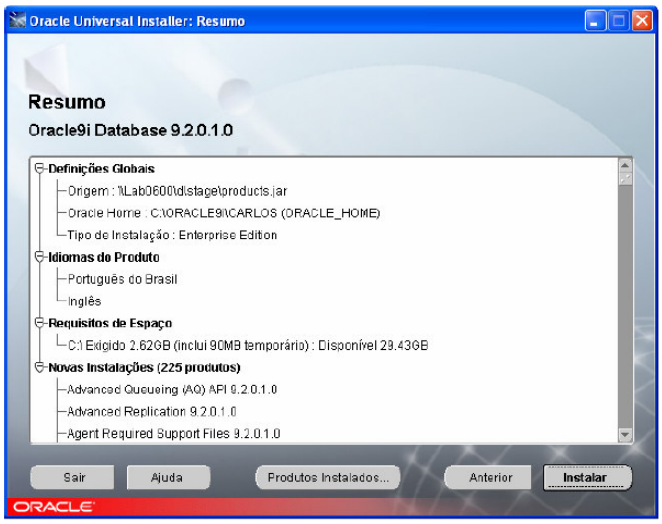


Figura 5.12: Tela do resumo das opções selecionadas.

Após o término do processo de instalação do Banco de Dados, a tela apresentada na Figura 5.13 será exibida. Nesse passo da configuração do Banco de Dados devem ser informadas as senhas para os usuários SYSTEM e SYS. Essa senha refere-se ao usuário que tem permissão de acesso livre sem restrições ao Banco de Dados. O usuário SYSTEM foi utilizado durante toda a configuração do Banco de Dados Distribuído, para o acesso ao editor *SQL-PLUS* e à ferramenta *Enterprise Manager Console.*

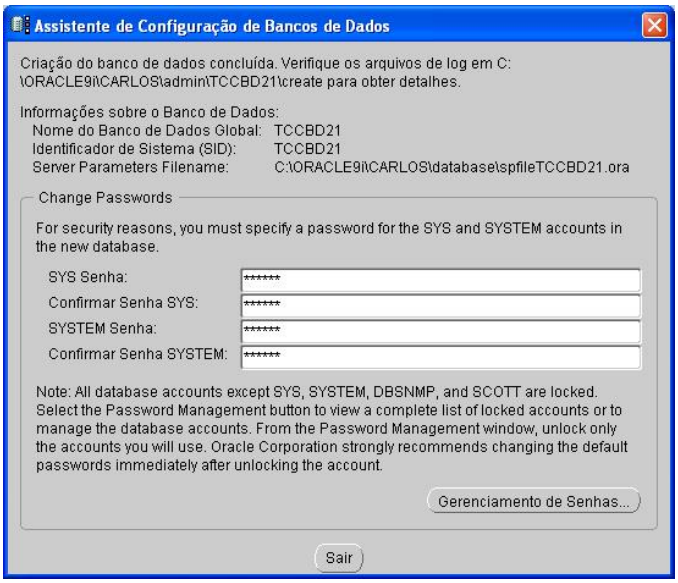


Figura 5.13: Tela de definição das senhas dos usuários SYSTEM E SYS.

5.2. Configuração do SGBD Oracle para operar de forma distribuída

Algumas características particulares como a configuração do *Listener, Métodos de Nomeação, Nome do Serviço de Rede* e criação do *Vínculo* do banco de dados devem ser definidas a fim de que o SGBD Oracle opere de forma distribuída.

6.2.1 Configuração do Listener

O *Listener* (ouvinte) é um processo que espera por conexões do cliente para acessar uma ou diversas bases de dados (Oracle, 2006). Para configurá-lo foi necessário acessar a ferramenta *Net Maneger* e mudar a porta de atendimento do *Listener* para a porta 1080, pois já havia nas máquinas um banco de dados instalado anteriormente. Em seguida, através da ferramenta *Net Configuration Assistent* foi criado um novo *Listener* para atender na porta 1521. Na Figura 5.14 é mostrada a tela inicial de configuração.

Figura 5.14: Tela inicial de configuração, inicialmente do Listener.

Por meio da tela mostrada na Figura 5.15, foi possível adicionar um novo *Listener*, nomeando-o conforme consta na Figura 5.16.

Figura 5.15: Segundo passo para a configuração de um Listener.

Figura 5.16: Tela para denominação do novo Listener.

O protocolo para comunicação e a porta a ser utilizada pelo *Listener* foram definidos por meio das telas mostradas na Figura 5.17 e Figura 5.18, respectivamente.

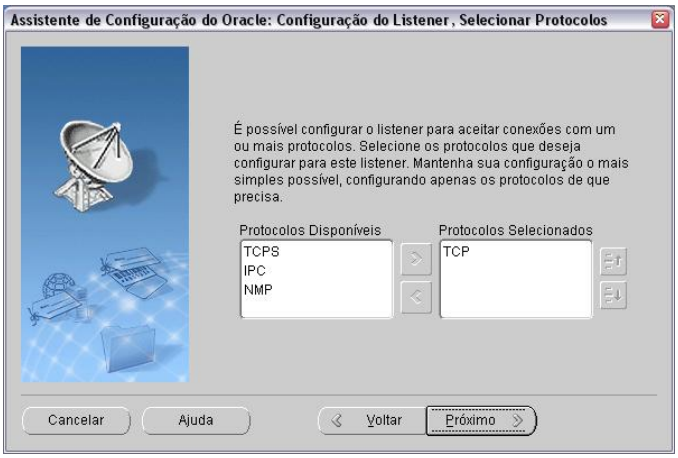


Figura 5.17: Tela de configuração do novo Listener.

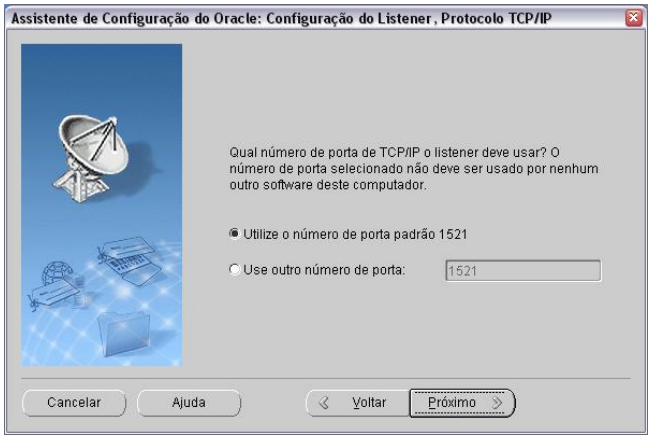


Figura 5.18: Tela de escolha do número da porta para a configuração do Listener.

Na Figura 5.19 e na Figura 5.20 são mostradas as telas de finalização da configuração.

Figura 5.19: Tela de configuração do Listener.

Figura 5.20: Tela onde há a informação que o Listener foi configurado com sucesso.

5.2.2 Configuração do Método de Nomeação

Prosseguindo com a instalação será configurado o *Método de nomeação*, foi escolhida a segunda opção, como mostrada na Figura 5.21.

Figura 5.21: Tela inicial para a configuração dos Métodos de Nomeação.

Existem cinco métodos de nomeação: *Nome do host, Local, Oracle Names, Sun NIS, DCE CDS.*

Nesse trabalho foi escolhido *Nome do host* e *Local*. Isso faz com que precisa-se criar e manter um arquivo de configuração dos nomes locais denominado *tnsnames.ora*. A conexão é feita através do nome do *host* das máquinas. Com esse método é preciso instalar o protocolo TCP/IP tanto no cliente como no servidor. Na Figura 5.22 é mostrada a configuração do *Método de Nomeação.*

Figura 5.22: Tela de configuração dos Métodos de nomeação.

Escolhendo a opção prosseguir, será finalizado a configuração dos *Métodos de nomeação*, exatamente como mostrada na Figura 5.23.

Figura 5.23: Tela de configuração de métodos de nomeação concluída.

5.2.3 Configuração dos Nomes de Serviços

O próximo passo é a configuração dos Nomes de serviços. Eles são responsáveis em prover a conectividade para sistemas computacionais distribuídos e heterogêneos. Para isso é

preciso escolher a terceira opção da tela inicial de configuração, como mostrada na Figura 5.24.

Figura 5.24: Tela de configuração do nome de serviços da rede.

Selecionando a opção *Próximo*, aparecerá uma tela (Figura 5.25) que permite adicionar um *Nome de Serviço*.

Figura 5.25: Tela de opções de ações para a configuração do nome de serviços da rede.

A próxima tela é da escolha da versão do banco de dados que deseja-se acessar. No caso, foi selecionada a primeira opção, como mostrada na Figura 5.26.

Figura 5.26: Tela de escolha da versão do banco de dados que será acessado.

Prosseguindo a instalação, será mostrada uma tela como na Figura 5.27. Esse passo solicita o nome do serviço que está sendo configurado.

Figura 5.27: Tela para informarmos o nome do serviço.

Em seguida é preciso escolher o protocolo de comunicação, exatamente como na Figura 5.28.

Figura 5.28: Tela de escolha do protocolo de comunicação.

Na próxima etapa deve-se informar o nome do *host* remoto, onde o banco de dados local irá se conectar, e a porta de comunicação. A porta deve ser a mesma informada na configuração do *Listener* feita anteriormente. Na Figura 5.29 é mostrada essa etapa da configuração.

Figura 5.29: Tela para informar o nome do host e porta.

Na próxima tela aparece uma pergunta para definir se a configuração será testada, como mostrada na Figura 5.30.

Figura 5.30: Tela de teste de conexão.

Se a configuração realizada estiver correta, a próxima tela mostrada será exatamente como na Figura 5.31.

Figura 5.31: Tela da mensagem do teste de conexão.

Escolhendo-se a opção *Próximo*, deverá ser informado o nome do serviço de rede configurado, como mostrada na Figura 5.32.

Figura 5.32: Tela da denominação do serviço de rede.

Prosseguindo, na próxima tela, demonstrada na Figura 5.33, é questionado se será configurado outro nome de serviço de rede.

Figura 5.33: Pergunta se queremos configurar outro serviço de rede.

5.2.4. Criação dos vínculos do banco de dados

O próximo passo para a finalização da configuração do banco de dados distribuído Oracle é a criação de um vínculo. A configuração do vínculo é a mais importante em Sistemas de Banco de Dados Distribuídos, pois é responsável pela comunicação entre os

servidores e clientes de forma transparente, fornecendo a sensação de que os usuários estão utilizando uma base de dados local.

Para configurá-lo é preciso usar a ferramenta *Enterprise Manager Console* e selecionar o banco de dados que foi configurado em passos anteriores. Com isso será fornecida uma tela para informar o *Login e Senha* do usuário (Figura 5.34). Em seguida, se os dados estiverem corretos serão fornecidos vários itens do lado esquerdo. Deve-se selecionar o item *Distribuído* e escolher a opção *Criar*, como mostrada na Figura 5.3. Então, será disponibilizada uma tela como mostrada na Figura 5.36.

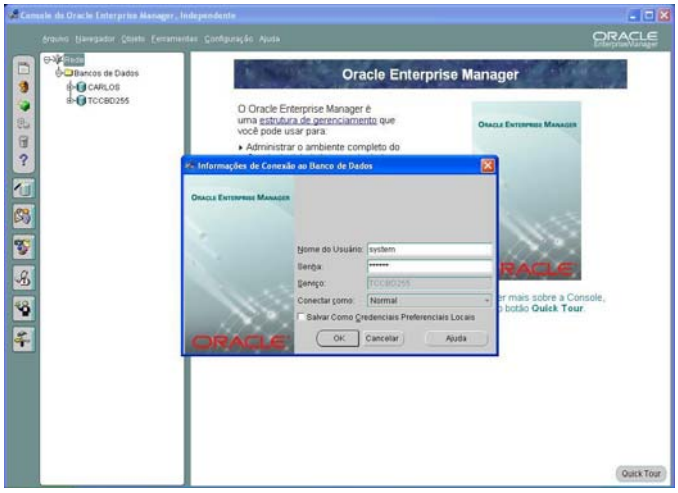


Figura 5.34: Tela de Login e Senha para a Criação do vínculo do banco de dados.

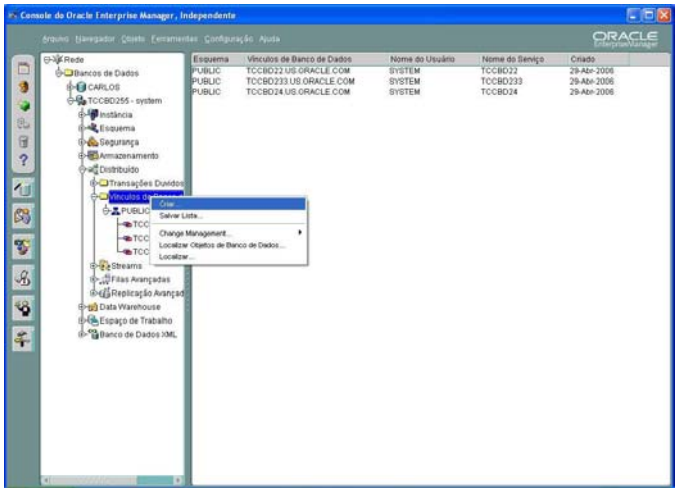


Figura 5.35: Menu para a Criação do vínculo do banco de dados.

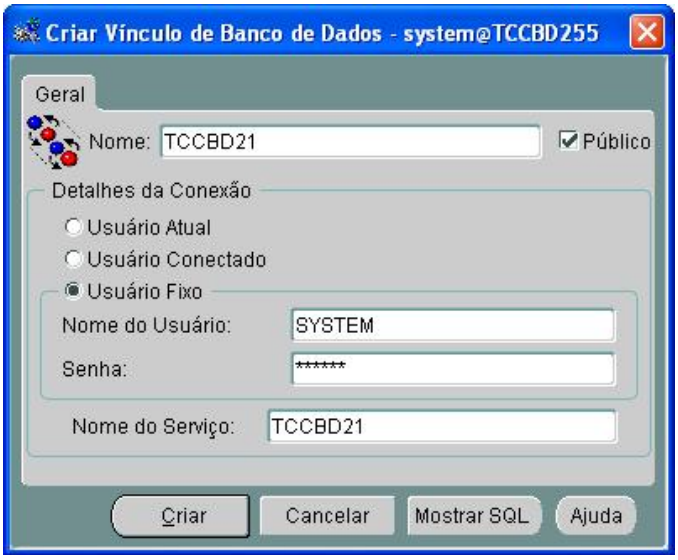


Figura 5.36: Criação do vínculo do banco de dados que será acessado remotamente.

Na tela reproduzida na Figura 5.36 é preciso informar o usuário e a senha que deverão ser utilizados para a comunicação com o Banco de Dados remoto. No campo nome deve ser informado o mesmo nome do Banco de Dados global. A opção *public* disponibiliza o *link* para todos os usuários que se conectarem no sistema. Caso esteja desmarcado, ele será privado, sendo que somente o usuário que o criou tem acesso a ele.

Esse procedimento deve ser repetido para configurar todas as máquinas. Na Figura 5.37 é mostrada a configuração da máquina LAB0625 com todos os vínculos criados.

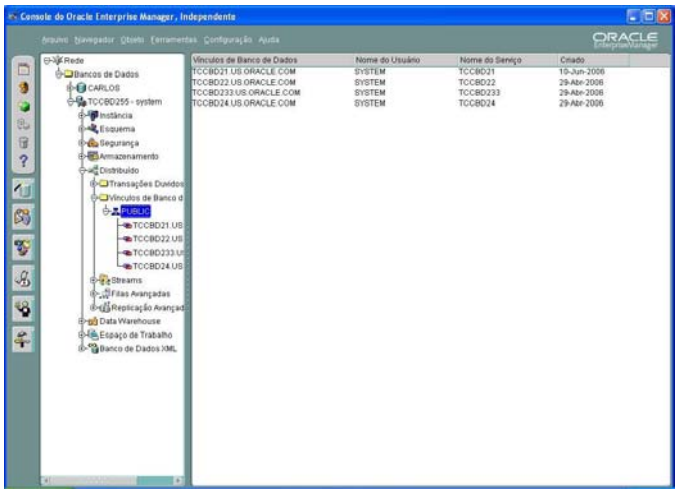


Figura 5.37: Todos os vínculos criados na máquina LAB0625 mostrados pela ferramenta Enterprise Manager Console.

Com o término da configuração do Vínculo do banco de dados, este está apto para ser usado de forma distribuída.

5.3. Definição dos Testes

Após a instalação do Oracle, as tabelas do modelo do banco de dados foram geradas a partir de um *script*, que foi executado em cada máquina separadamente (Anexo A no final deste trabalho).

Em seguida, estabeleceram-se os testes necessários a fim de atingir os objetivos propostos. Para isso foi confeccionado um programa, como citado anteriormente, utilizando a linguagem de programação Java.

Antes de iniciar qualquer teste foi criado um sinônimo a cada tabela pertencente a cada máquina para se obter a transparência de localização. A sintaxe da criação de um sinônimo está na Figura 5.38.

```
CREATE SYNONYM <NomedoSinonimo>
FOR <objetoRemoto@nomedoBancodeDadosGlobal>
```

Figura 5.38: Sintaxe da SQL para a criação de sinônimos.

Foram realizados testes de consistência e desempenho. A idéia básica com relação aos testes de desempenho é demonstrada pela Figura 5.39. Basicamente os testes realizados consistiram em inserir uma grande quantidade de registros no BD utilizando como interface a linguagem Java e verificar como o SGBD se comporta com a inserção, recuperação e exclusão de diferentes quantidades de dados em diferentes tabelas.

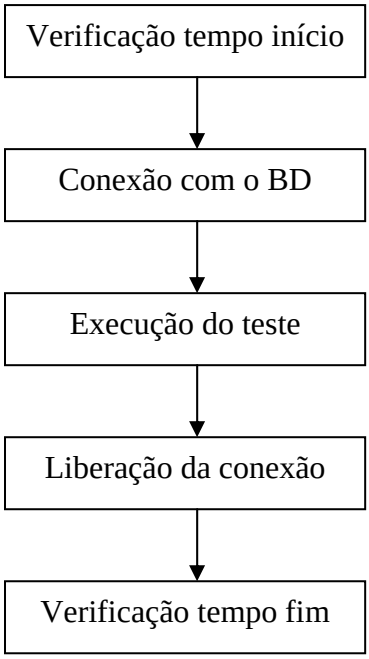


Figura 5.39: Esquema da execução dos testes.

5.3.1. Testes de Integridade e Consistência

Esse primeiro teste foi feito na ferramenta *SQL PLUS* para a verificação de integridade e consistência dos dados. Tinha o objetivo de verificar se o modelo de banco de dados elaborado era adequado para os demais futuros. Consistiu em verificar a garantia de integridade e consistência dos dados, da seguinte maneira:

1) Na tabela Cidade há um campo de nome UF, do tipo *varchar2*, com tamanho 3, para o qual há uma verificação a fim de garantir a inserção das seguintes cadeia de caracteres:: “SP”, “MG”, “RJ”, “ES”, “PR”, “SC”, “RS”. O teste de integridade e consistência consistiu em inserir na tabela uma cidade cujo valor estivesse fora da especificação projetada no banco de dados.

2) O segundo teste foi para verificar a integridade referencial das tabelas. Para isso foi feito um teste inserindo registro na tabela filha (por exemplo, a tabela Modelo) que não tinha correspondência com a pai (por exemplo, a tabela *Montadora*). Em seguida, outro teste realizado foi a remoção de registros da tabela pai (*Montadora*) que tinham registros filhos correspondentes na tabela filha. Um último teste realizado foi a tentativa de inserir dois ou mais registros com a mesma chave primária em qualquer uma das tabelas.

5.3.2. Estabelecendo a conexão

Na Figura 5.40 é mostrado o código implementado para a conexão com o banco de dados. Foi preciso utilizar o pacote oracle.jdbc.util para instanciar objetos de classes do Oracle, utilizando a instrução *import* seguido do nome do pacote, juntamente com o nome da classe a ser utilizada. A conexão é feita passando o nome do *host*, número da porta, e a senha para o método *getConnection()* da classe *DriverManager*.

O método *forName()* tenta carregar o *driver* passando como parâmetro. Caso não consiga, será gerada uma exceção do tipo *ClassNotFoundException*.

O método *getConnection()* da classe *DriverManager* tentará estabelecer a conexão. Retorna um objeto do tipo *Connection* caso a conexão seja bem sucedida. Caso contrário, retorna uma exceção do tipo *SQLException.* A instância do objeto *Connection* retornada por *getConnection()* é atribuída à referência *conn* do tipo *Connection*. O usuário passado para *getConection()* deve estar criado no banco de dados e ter permissão de acesso ao banco de dados.

```
Import oracle.jdbc.util.*;
…
 try {   String driverName = "oracle.jdbc.driver.OracleDriver";
          Class.forName(driverName);      }
          catch (ClassNotFoundException e)
          { //retorna o erro e pára
          jTextArea1.append("Não pode carregar o drive: "+e);
         }
       try
       {   String serverName = "lab0621";
          String portNumber = "1521";
          String sid = "TCCBD21";
          String url = "jdbc:oracle:thin:@" + serverName + ":" + portNumber + ":" + sid;
          String username = "SYSTEM";
          String password = "vr1326";
          con = DriverManager.getConnection(url, username, password);
       }  catch (Exception e)
          {   jTextArea1.append("Não pode conectar: "+e);    }
```

Figura 5.40: Trecho de código que estabelece a conexão com o BD Oracle.

5.3.3. Testes de Inserção dos dados

Todos os testes descritos desse tópico em diante foram executados por meio de um programa elaborado em Java, a partir da máquina denominada LAB0621, ou seja, essa máquina nos testes feitos será considerada como local e as outras como remotas.

Os testes de inserção consistiram em inserir os dados nas diversas tabelas do modelo de banco de dados, considerando quantidades variáveis de registros, a fim de estabelecer comparações de desempenho. Foram utilizadas as quantidades de 50, 500, 5.000 e 50.000 registros. Para cada inserção era registrado o tempo inicial e final. Na Figura 5.41 é mostrado um trecho do programa para inserir registros na tabela cliente.

```
        Try {
          jTextArea1.append("O numero de Registro para inserção é: "+QtReg+"\n");
          Statement stmt= connection.createStatement();
          long timestamp=System.currentTimeMillis();
          double soma;
          float time2;
          QtReg= jTextField1.getText();
          v=1;
          soma=v+Integer.parseInt(QtReg); //transformar string em integer.
          int SomaCount = 0;
          long inicio=System.currentTimeMillis();
           while ((v+1) <= soma)
            {
               int res= stmt.executeUpdate(VOpcao+" into teste5 values ('RUA,222', '3333-
3333', 'Vanessa', "+v+")");
              v+=1;
            }
         long fim=System.currentTimeMillis();
          long t= fim - inicio;
          time2= t/1000; //para achar em segundos o tempo
          jTextArea1.append("\nO tempo gasto para Inserção do(s) dado(s) na maquina
Lab0625 foi: "+time2+" segundos\n");
          connection.commit();
```

Figura 5.41: Trecho do programa para inserir os dados na tbl. Cliente na máquina lab0625.

O retorno da contagem de tempo é em segundos, registrado pelo programa confeccionado. Foi gerado um arquivo ASCII com os tempos de respostas de cada inserção em cada máquina. Uma observação a ser feita é que a geração deste arquivo não interfere na contagem de tempo, pois ele é criado antes da primeira inserção e após a última inserção. A Figura 6.42 é mostrado um trecho do arquivo gravado.

```
O numero de Registro para inserção é: 50

O tempo gasto para Inserção do(s) dado(s) na maquina Lab0625 foi: 4.0
segundos

O tempo gasto para Inserção do(s) dado(s) na maquina lab0624 foi: 9.0
segundos

O tempo gasto para Inserção do(s) dado(s) na maquina lab0623 foi: 13.0
segundos

O tempo gasto para Inserção do(s) dado(s) na maquina Lab0622 foi: 4.0
segundos

O tempo gasto para Inserção do(s) dado(s) na maquina lab0621 foi: 0.0
segundos
```

Figura 5.42: Exemplo do retorno gerado pelo programa no arquivo ASCII.

5.3.4. Testes de Recuperação dos dados

Os testes de recuperação consistiram em buscar os dados nas diversas tabelas do modelo de banco de dados. Os resultados também foram gravados em um arquivo ASCII com os tempos de respostas de cada busca em cada máquina e os dados recuperados das tabelas. Na Figura 5.43 é mostrado um trecho do programa para a recuperação dos dados na tabela cliente.

```
        try {
          jTextArea1.append("O numero de Registro para seleção é: "+QtReg+"\n");
          Statement stmt= connection.createStatement();
          long timestamp=System.currentTimeMillis();
          double time,time2, soma;
          File outputFile = new File("Resposta.txt");
          FileWriter out = new FileWriter(outputFile);
          long inicio=System.currentTimeMillis();
           ResultSet rs= stmt.executeQuery(VOpcao+" * from teste5");
          while (rs.next()){
            //CLIENTE
            out.write(""+ "Endereço:"+rs.getString("Endereco")+"\n" + "");
            out.write(""+ "Telefone:"+rs.getString("Telefone")+"\n" + "");
            out.write(""+ "Nome:"+rs.getString("Nome")+"\n" + "");
            out.write(""+ "CPF:"+rs.getDouble("CPF")+"\n\n" + "");
          }
          long fim=System.currentTimeMillis();
          rs.close();
           long t= fim - inicio;
          DecimalFormat decimal = new DecimalFormat( "0.00" );
          time2= t/1000; //para achar em segundos o tempo
          decimal.format(time2);
          jTextArea1.append("\nO tempo gasto para Recuperação do(s) dado(s) na
maquina Lab0625 foi: "+time2+" segundos\n");
          jTextArea1.append("\nRegistros selecionados com sucesso na tabela");
         stmt.close();
         out.write(""+ jTextArea1.getText() + "");
          out.close();
          }catch (Exception e)  {
          //retorna o erro e pára
          JOptionPane.showMessageDialog(null,"Impossível Recuperar os Dados, pelo
Seguinte Motivo:\n"+e,"Atenção",JOptionPane.WARNING_MESSAGE);
          }finally {
          }
```

Figura 5.43: Trecho do programa para a recuperação dos dados na tabela Cliente na máquina lab0625.

Na Figura 5.44 é mostrado o arquivo gerado pelo programa.

```
Codigo Conc.:1.0
Codigo Mont.:1.0
Tel.:3.3333333E7
Tipo Tel.:R
Codigo Conc.:2.0
Codigo Mont.:2.0
Tel.:3.3333333E7
Tipo Tel.:R
Codigo Conc.:3.0
Codigo Mont.:3.0
Tel.:3.3333333E7
Tipo Tel.:R

O numero de Registro para seleção é: 50
O tempo gasto para Recuperação do(s) dado(s) na maquina Lab0625 foi: 9.0
segundos
O tempo gasto para Recuperação do(s) dado(s) na maquina Lab0624 foi: 4.0
segundos
O tempo gasto para Recuperação do(s) dado(s) na maquina Lab0623 foi: 4.0
segundos
O tempo gasto para Recuperação do(s) dado(s) na maquina Lab0622 foi:
9.0 segundos
O tempo gasto para Recuperação do(s) dado(s) na maquina Lab0621 foi: 0.0
segundos
Registros selecionados com sucesso na tabela
```

Figura 5.44: Exemplo do retorno gerado pelo programa no arquivo ASCII.

5.3.5. Testes de Exclusão dos dados

Os testes de exclusão consistiram em excluir os dados das tabelas do modelo de banco de dados. Da mesma forma, foram registrados o tempo de início e fim da operação em um arquivo ASCII. Na Figura 5.45 é mostrado um trecho do programa onde são excluídos registros da tabela cliente na máquina LAB0625.

```
 try {
         jTextArea1.append("O numero de Registro para deletar é: "+QtReg+"\n");
         //recebe um statement de conexao
         Statement stmt= connection.createStatement();
         long timestamp=System.currentTimeMillis();
         double time,time2, soma;
         long inicio=System.currentTimeMillis();
         ResultSet rs= stmt.executeQuery(VOpcao+" from teste5");
         long fim=System.currentTimeMillis();
         jTextArea1.append("Tempo inicio:"+inicio+"\n");
         jTextArea1.append("Tempo final:"+fim+"\n");
         long t= fim - inicio;
         DecimalFormat decimal = new DecimalFormat( "0.00" );
         time2= t/1000; //para achar em segundos o tempo
         decimal.format(time2);
         jTextArea1.append("\nO tempo gasto para Apagar o(s) dado(s) na maquina Lab0625
foi: "+time2+" segundos\n");
         jTextArea1.append("\nRegistros selecionados com sucesso na tabela");
        connection.commit();
         stmt.close();
         jTextArea1.append("\nRegistros apagados com sucesso na tabela");
         File outputFile = new File("Resposta.txt");
         FileWriter out = new FileWriter(outputFile);
         out.write(""+ jTextArea1.getText() + "");
         out.close();
        } catch (Exception e)  {
            JOptionPane.showMessageDialog(null,"Impossível Deletar os Dados, pelo
Seguinte Motivo:\n"+e,"Atenção",JOptionPane.WARNING_MESSAGE);          }
         finally{ }
```

Figura 5.45: Trecho do programa para a exclusão dos dados na tabela Cliente na máquina lab0625.

Na Figura 5.46 é mostrado um trecho do arquivo gerado após execução da exclusão dos registros.

```
O numero de Registro para a exclusão é: 50
O tempo gasto para Apagar o(s) dado(s) na maquina Lab0625 foi: 9.0
segundos
O tempo gasto para Apagar o(s) dado(s) na maquina Lab0624 foi: 4.0
segundos
O tempo gasto para Apagar o(s) dado(s) na maquina Lab0623 foi: 4.0
segundos
O tempo gasto para Apagar o(s) dado(s) na maquina Lab0622 foi: 9.0
segundos
O tempo gasto para Apagar o(s) dado(s) na maquina Lab0621 foi: 0.0
segundos
Registros apagados com sucesso na tabela
```

Figura 5.46: Exemplo do retorno gerado pelo programa no arquivo ASCII.

6. RESULTADOS E DISCUSSÕES

Após a execução das fases descritas anteriormente e da verificação dos resultados obtidos muitas considerações podem ser levantadas a respeito do SGBD Oracle avaliado, apresentados a seguir.

6.1. Testes de Consistência e Integridade

No primeiro teste, que consistia em inserir dados fora da especificação projetada no banco de dados, no campo UF, retornou um erro informando que não foi possível inserir, pois o parâmetro passado estava fora das especificações.

Já no segundo teste, que era inserir um registro na tabela filha que não tinha correspondência na tabela pai, também retornou um erro informando que não foi possível inserir porque a referência da tabela pai não foi encontrada. E no teste da exclusão dos dados da tabela pai com registro correspondente na tabela filha, houve retorno de um erro informando que não foi possível excluir porque havia registros correspondentes na tabela filha.

Portanto, todos os testes de integridade e consistência do modelo do banco de dados provaram que o banco projetado atende às necessidades deste trabalho e que todos os dados que serão inseridos e excluídos possuem consistência.

6.2. Testes de Inserção dos Dados

Como descrito anteriormente, os testes de inserção foram feitos com quantidades de registros especificados, que foram 50 registros, 500 registros, 5.000 registros e 50.000 registros inseridos em cada máquina. Todas as tabelas do modelo do banco de dados foram testadas.

6.2.1. Tabelas com um Relacionamento

As tabelas que possuem somente um relacionamentos são:

- Cliente
- Cidade
- Telefone

Nos testes de inserção a máquina LAB0623 gastou um tempo maior de inserção do que as outras máquinas remotas (LAB0622, LAB0624 E LAB0625). Foi verificada cada configuração da máquina e a única alteração entre elas foi a marca do HD da máquina Lab0623 que é SAMSUNG e as outras são WESTERN DIGITAL, porém, as especificações dos dois modelos de HD possuem as mesmas características, como: mesma rotação (7.200 rpm), capacidade de 80GB e tamanho do Buffer 2MB, entre outros. Então foi observado o espaço ocupado em cada HD. Na máquina LAB0623 o HD está com 21,3 GB ocupados e nas outras máquinas está com 19 GB ocupados.

Pela comparação dos testes nessas tabelas, pode-se verificar que as tabelas *Cidade* e *Telefone* possuem um tempo maior de inserção em todas as máquinas. Essas tabelas se diferenciam da tabela *Cliente* nos campos UF (tabela Cidade) e Tipo_do_Telefone (tabela Telefone) pois possuem uma restrição de *CHECK* que foi criado no desenvolvimento da modelagem do banco de dados, onde especifica que os valores válidos para o campo UF são: SP, MG, RJ, ES, PR, SC, RS e para o campo Tipo_do_Telefone são: C (celular), F (fixo), T (trabalho), X (fax) e R (recado). O *script* da criação dessas tabelas e das outras pode ser visto no Anexo A ao final desse trabalho.

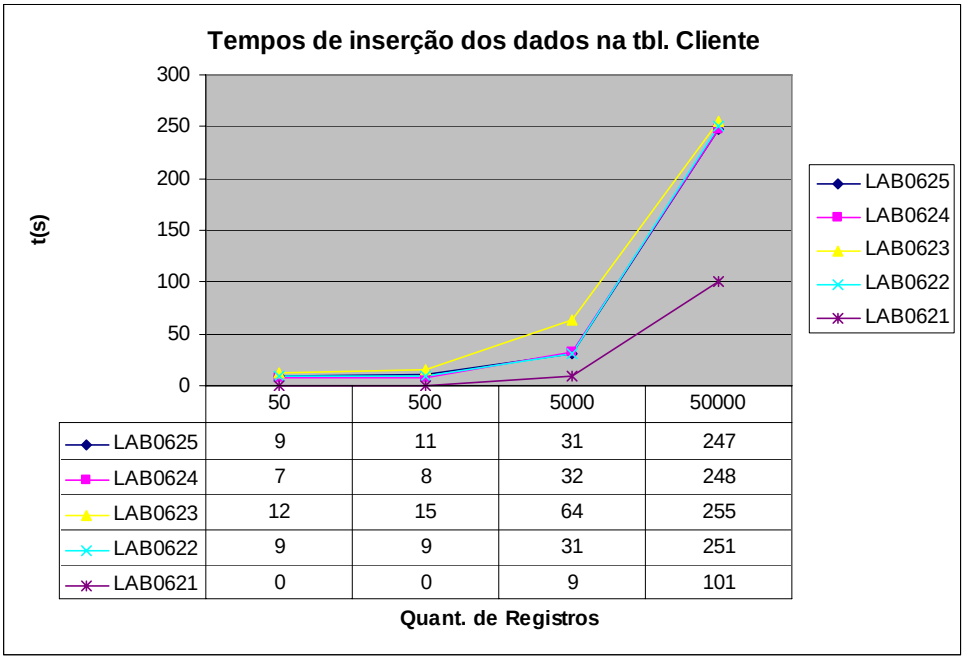
Apesar da tabela *Cliente* possuir mais atributos que a tabela *Cidade*, os tempos de inserção de registro da menor tabela, considerando a quantidade de atributos, é maior. Pode-se, preliminarmente, deduzir que a quantidade de atributos não exerce tanta influência na performance do banco de dados Oracle. E que a influência maior está na diferença de

estrutura, pois o uso da cláusula *CHECK* na criação da tabela faz com que em cada inserção o SGBD verifique se o campo atende às especificações ou não, aumentando o tempo requerido de inserção. As tabelas *Cidade* e *Telefone* apresentaram quase os mesmos tempos, portanto, a quantidade de campos que compõem a chave primária, para o Oracle, também não altera o tempo comparado a uma tabela que tenha somente um atributo na chave primária.

A respeito de inserção de dados em base local ou remota, o tempo é bem diferente quando comparados. É bem menor o tempo de inserção local do que o remoto, como seria esperado.

Uma grande alteração de tempo ocorre também ao se comparar a quantidade de registros inseridos, ou seja, o tempo geral de inserção de 50 registros é 2827,45% superior comparado à quantidade de 50.000 registros.

Nas Figuras 6.1, 6.2 e 6.3 são mostrados os resultados dos testes de inserção, respectivamente nas tabelas *Cliente*, *Cidade e Telefone*.



	50	500	5000	50000
LAB0625	9	11	31	247
LAB0624	7	8	32	248
LAB0623	12	15	64	255
LAB0622	9	9	31	251
LAB0621	0	0	9	101

Figura 6.1: Comparação de inserção na tabela Cliente.

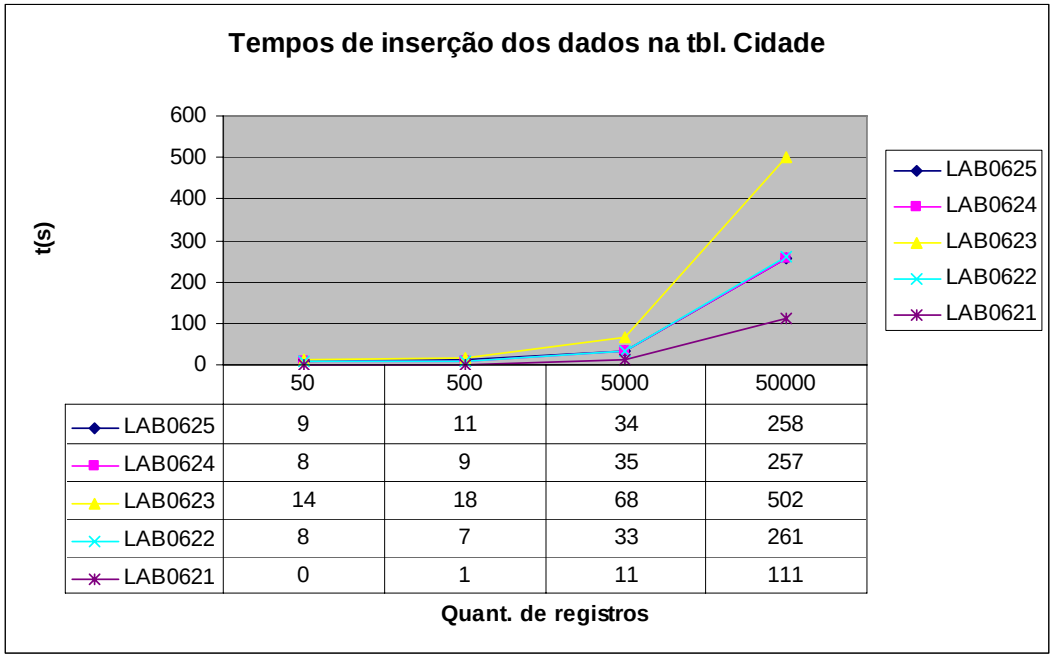


Figura 6.2: Comparação de inserção na tabela Cidade.

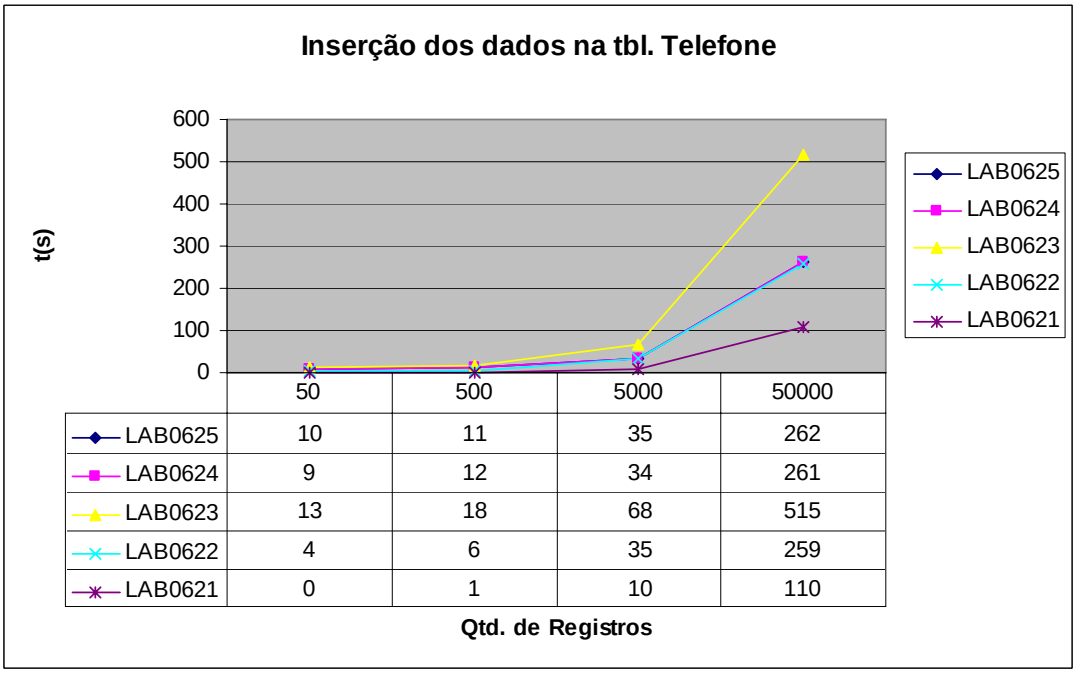


Figura 6.3: Comparação de inserção na tabela Telefone.

6.2.2. Tabelas com mais de um Relacionamento

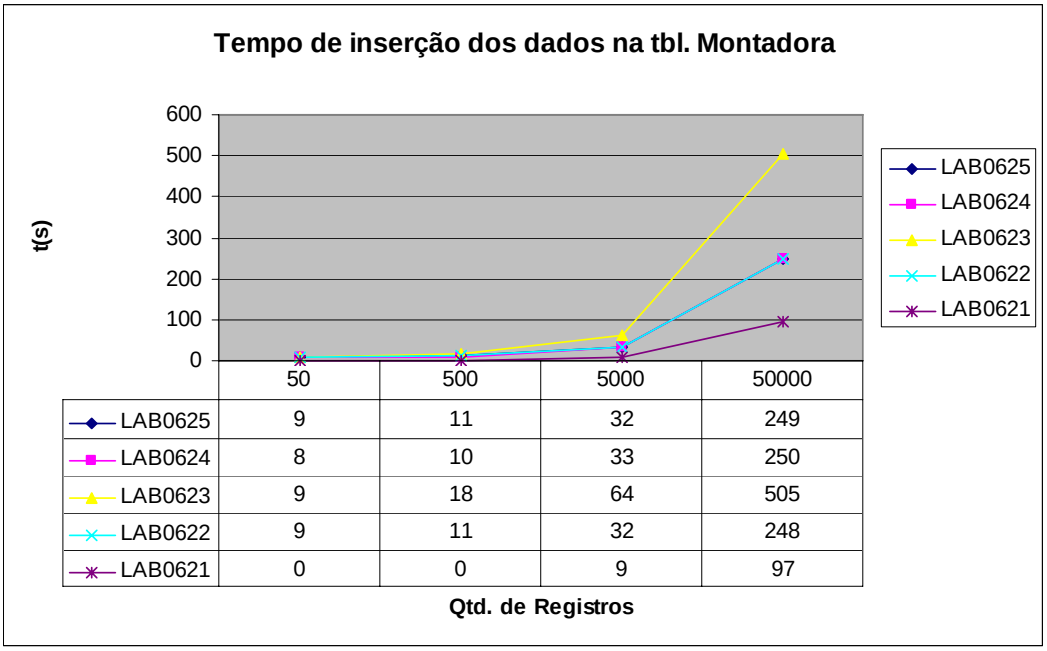
As tabelas que possuem mais do que um relacionamento são:

- Montadora, com dois relacionamentos.
- Modelo, com três relacionamentos.
- Concessionária, com quatro relacionamentos.

As tabelas restantes serão especificadas em um outro tópico, pois os testes foram executados com uma pequena diferença.

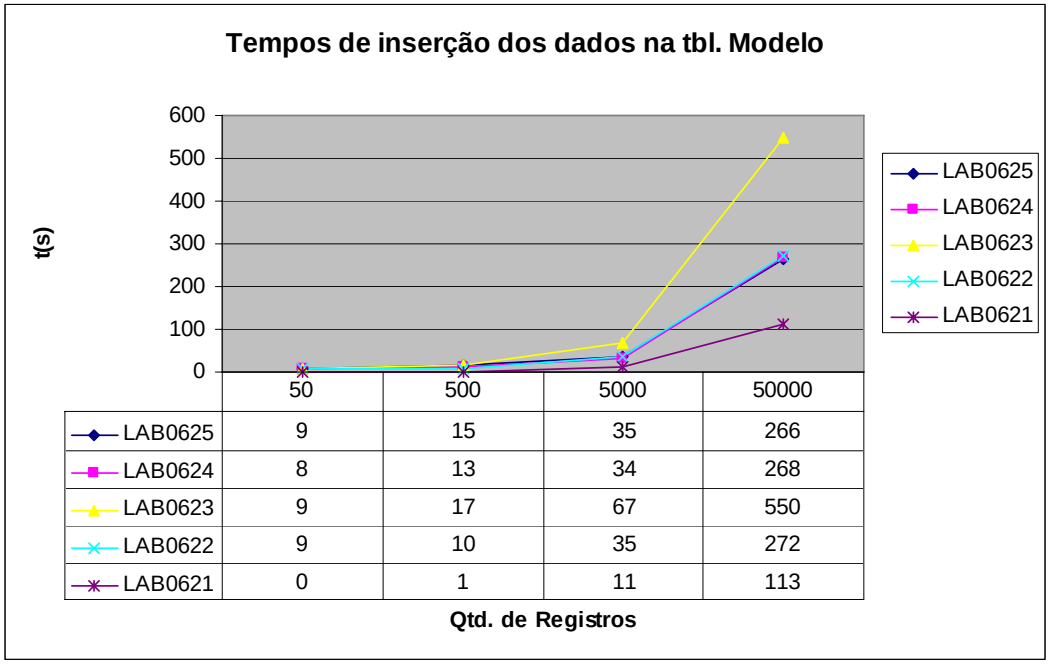
Nos testes dessas tabelas também houve um tempo maior para a máquina LAB0623. Quando comparada com os restantes das máquinas, o seu tempo fica quase dobrado.

As diferenças nos tempos podem ser vistas nas Figuras 6.4, 6.5 e 6.6 a seguir.



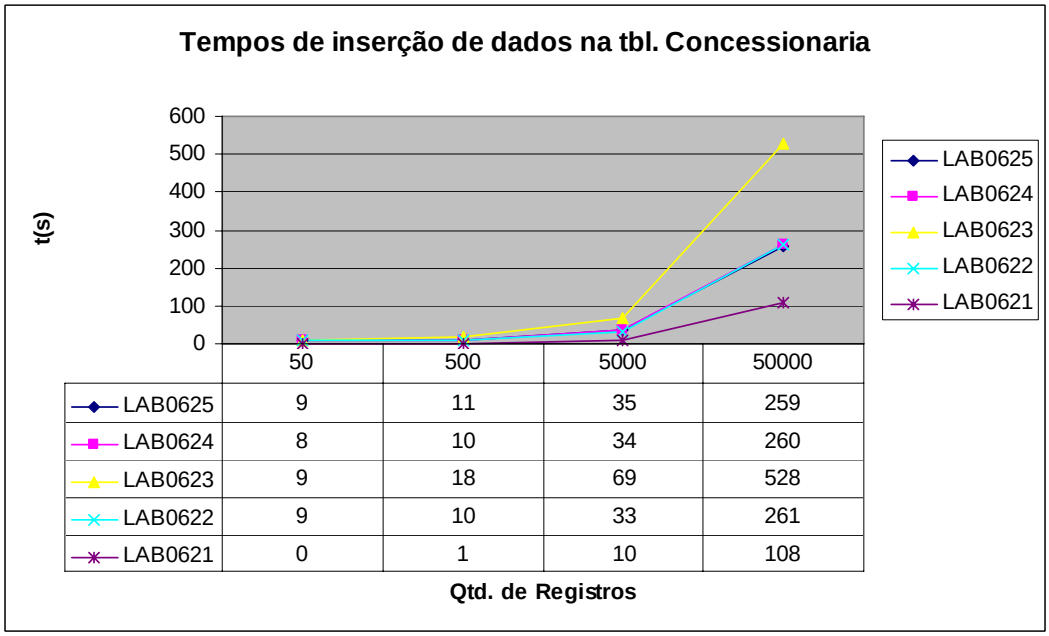
	50	500	5000	50000
LAB0625	9	11	32	249
LAB0624	8	10	33	250
LAB0623	9	18	64	505
LAB0622	9	11	32	248
LAB0621	0	0	9	97

Figura 6.4: Comparação de inserção na tabela Montadora.



	50	500	5000	50000
LAB0625	9	15	35	266
LAB0624	8	13	34	268
LAB0623	9	17	67	550
LAB0622	9	10	35	272
LAB0621	0	1	11	113

Figura 6.5: Comparação de inserção na tabela Modelo.



	50	500	5000	50000
LAB0625	9	11	35	259
LAB0624	8	10	34	260
LAB0623	9	18	69	528
LAB0622	9	10	33	261
LAB0621	0	1	10	108

Figura 6.6: Comparação de inserção na tabela Concessionária.

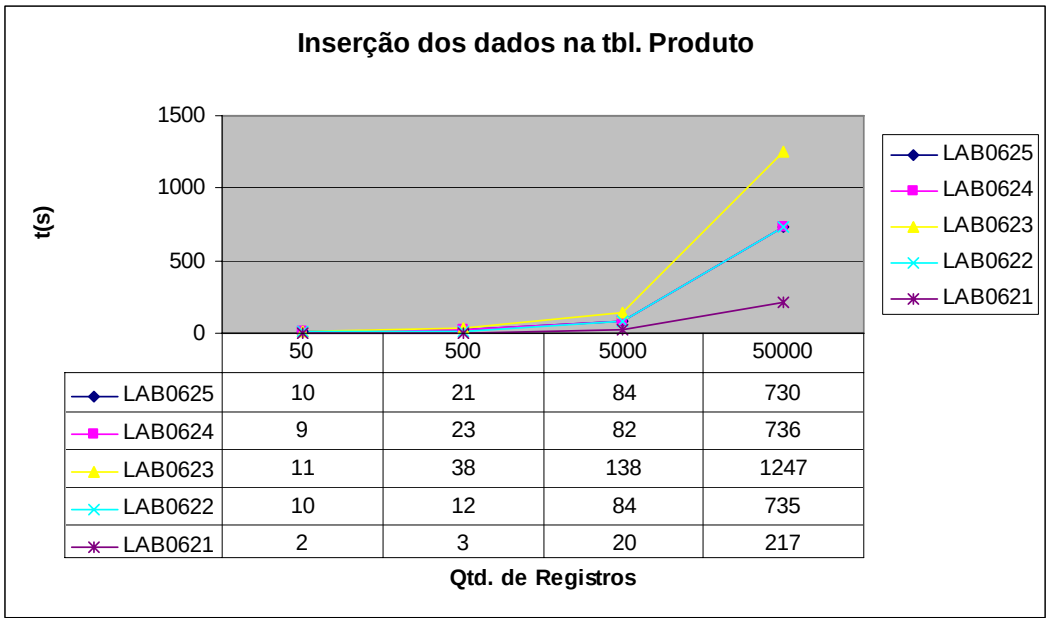
A quantidade de inserção de registros continua influenciando nos tempos, ou seja, quanto mais registros se insere menos tempo se leva para inserir. Por exemplo, na tabela *Montadora*, o tempo de inserção de 50 registros é 2641,50 % maior que o tempo de inserção de 50.000 registros; na tabela *Modelo* essa mesma comparação das quantidades de inserção de 50 registros é 2413,79% maior que a quantidade de 50.000 registros; e por último na tabela *Concessionária*, a quantidade de 50 registros é 2641,50% maior que a quantidade de 50.000 registros.

O tempo, no geral, nas quantidades de 50.000 registros da tabela *Montadora* é 6,93% menor do que da tabela *Modelo* e *Concessionária*, porém, o tempo da tabela *Concessionária*, apesar de possuir mais relacionamentos que a *Modelo*, tem um tempo 0,47% menor que ela. Essa diferença pode ser explicada, pois na tabela Modelo o campo Tipo possui validações para ele, não podendo ser diferente de: S (*sedan*), H(*hatch*) e P(perua).

6.2.3. Tabelas com testes diferenciados

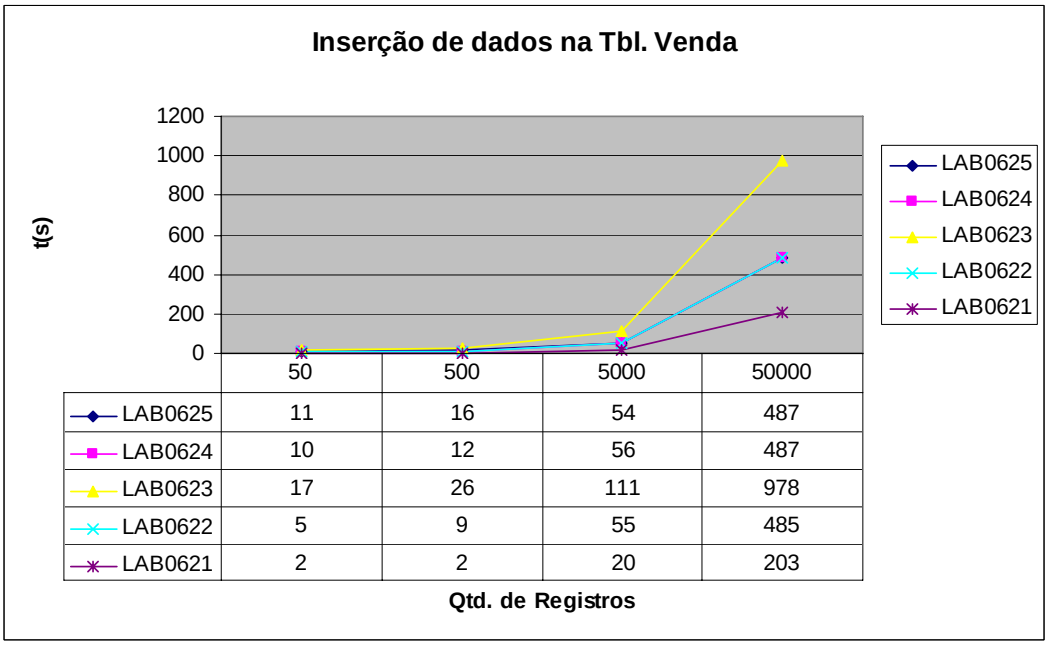
Nas Figuras 6.7 e 6.8 são ilustrados os tempos referentes aos testes dessa etapa.

Nessa etapa os dados foram inseridos em mais de uma tabela simultaneamente. Por exemplo, a tabela Produto tem uma especialização com as tabelas Veículo e Peça, onde essa última possui um relacionamento com a tabela Opcionais.



	50	500	5000	50000
LAB0625	10	21	84	730
LAB0624	9	23	82	736
LAB0623	11	38	138	1247
LAB0622	10	12	84	735
LAB0621	2	3	20	217

Figura 6.7: Comparação de inserção na tabela Produto.



	50	500	5000	50000
LAB0625	11	16	54	487
LAB0624	10	12	56	487
LAB0623	17	26	111	978
LAB0622	5	9	55	485
LAB0621	2	2	20	203

Figura 6.8: Comparação de inserção na tabela Venda.

Os testes de inserção foram feitos da seguinte maneira:

Ao inserir os dados na tabela *Produto*, se o campo *tipo_produto* fosse “V”esse inseriria um registro na tabela *Veículo* com a mesma descrição no campo *CodigoBarras*. Caso esse campo da tabela Produto fosse “P” ele inseriria um registro na tabela *Peça* e também na tabela *Opcionais* com a mesma descrição no campo *CodigoBarras*. Foi especificado que durante a inserção dos dados na tabela *Produto* das máquinas LAB0625, LAB0623 e LAB0621 o campo *tipo_produto* receberia o valor “V” e o restante das máquinas o valor “P”, ou seja, nos testes com o *tipo_produto* recebendo “V” eram inseridos dados em três tabelas diferentes, enquanto que com o mesmo campo recebendo o valor “P” os dados eram inseridos somente em duas tabelas distintas.

No caso da inserção dos dados na tabela *Venda* e *Itens_Venda* foi usado o mesmo procedimento, ou seja, ao inserir os dados na tabela *Venda* logo em seguida era inserido também um registro correspondente na tabela *Itens_Venda*, onde o campo *Numero_da_venda* era o que se repetia nas tabelas.

Esse procedimento foi adotado pois não existiriam informações nas tabelas filhas (tabelas: *Veículo, Peça, Opcionais e Itens_Venda*) se não houvesse dados correspondentes nas tabelas pai (*Produto e Venda*).

Nos testes feitos com essas tabelas, novamente aconteceu da máquina LAB0623 ter um tempo mais elevado que as outras máquinas.

Com base nos resultados foi verificado que a inserção de dados em base local realmente é mais rápida do que em base remota, chegado essa diferença em 326,31%.

Comparando os resultados somente na tabela de *Produtos,* onde o campo *Tipo_Produto* é igual a “V”, os tempos são 45% maiores do que quando o tipo é “P”. Isso pode ser explicado devido ao tempo elevado da máquina LAB0623 que tem seu tempo quase duas vezes maior.

Voltando à comparação entre os dois conjuntos de testes, que seriam as tabelas *Produto e Venda*, o tempo da tabela *Produto* foi superior ao da outra tabela, ou seja, quando se tem uma especialização o tempo de inserção aumenta ao ser comparado com uma tabela de relacionamento vários para vários, ou seja, vários registros na tabela *Venda* pode ter várias correspondências na tabela *Itens_Venda* e vice- versa. Também é preciso levar em conta que o campo *Tipo_Produto* na tabela *Produto* tem uma restrição de tipo de dados cujo, campo só aceita valores “P” (peça) ou “V” (veículo), podendo assim aumentar o tempo de inserção dos dados.

6.3. Testes de Recuperação de Dados

Todos os testes de recuperação de dados retornavam seus resultados diretamente em um arquivo ASCII, por meio do qual é possível a visualização dos dados recuperados das tabelas, juntamente com os dados da contagem do tempo da execução dessa operação.

6.3.1. Tabelas com um Relacionamento

Como dito anteriormente, as tabelas com um relacionamento são: *Cliente, Cidade e Telefone.*

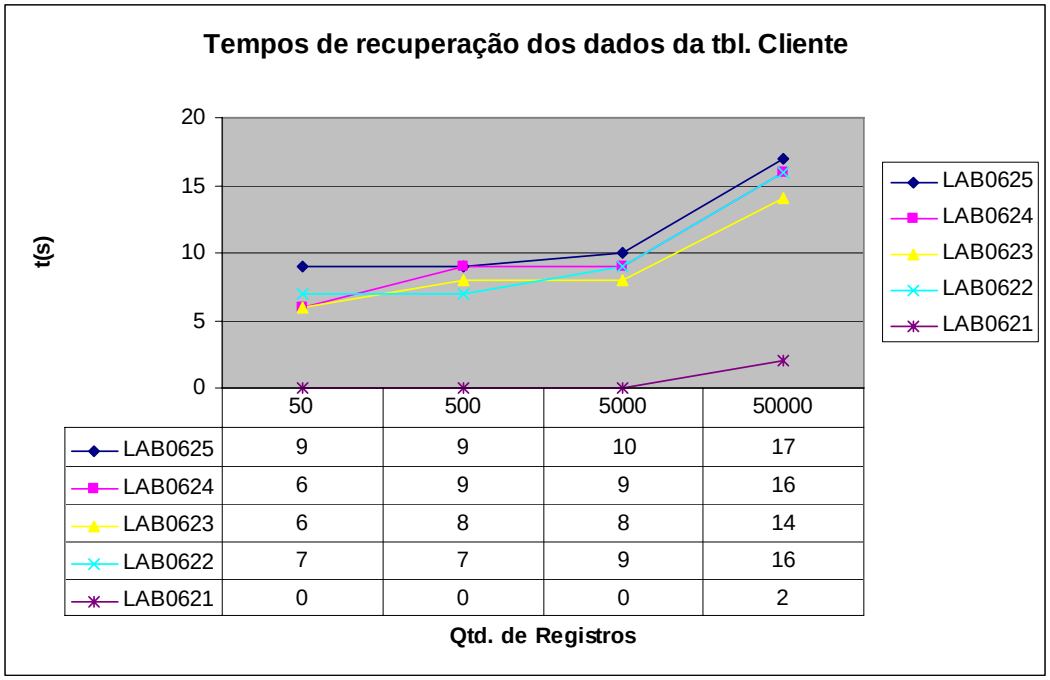
Nesse teste não houve grande diferença de tempo verificada nos testes de inserção, onde a máquina LAB0623 fica com o seu tempo maior que das outras máquinas.

Pode-se perceber que as diferenças entre os tempos das três tabelas são bem sutis. Os tempos de busca dos dados são quase os mesmos para as três tabelas, ou seja, no caso do teste

de recuperação dos dados as estruturas das tabelas não alteram em quase nada o tempo de execução da recuperação dos dados.

A influência maior que se pode observar é em relação à quantidade de registros - quanto mais dados se tem para buscar, menos tempo leva para buscá-los e também se os dados recuperados estão em uma base de dados local ou remota, localmente os tempos de resposta são bem menores que remotamente.

Os resultados deste teste podem ser vistos nas Figuras 6.9, 6.10 e 6.11.



	50	500	5000	50000
LAB0625	9	9	10	17
LAB0624	6	9	9	16
LAB0623	6	8	8	14
LAB0622	7	7	9	16
LAB0621	0	0	0	2

Figura 6.9: Comparação de recuperação dos dados na tabela Cliente.

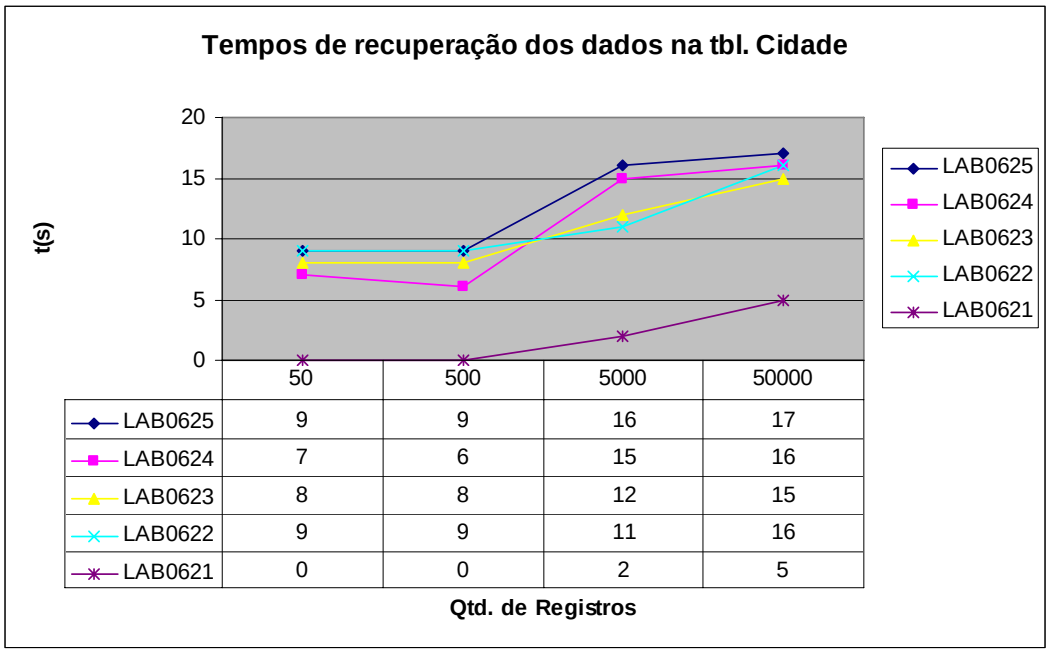


Figura 6.10: Comparação de recuperação dos dados na tabela Cidade.

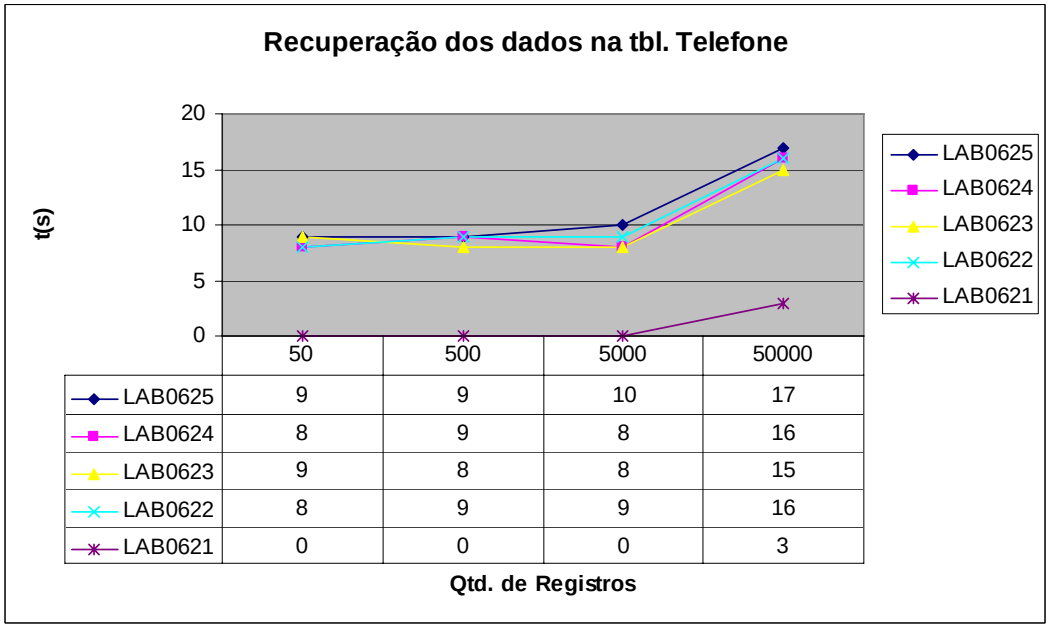


Figura 6.11: Comparação de recuperação dos dados na tabela Telefone.

6.3.2. Tabelas com mais do que um Relacionamento

As tabelas que possuem mais do que um relacionamento são: *Montadora* (dois relacionamentos), *Modelo* (três relacionamentos) e *Concessionária* (quatro relacionamentos).

Através dos resultados obtidos com este teste, é possível visualizar que não houve uma alteração evidente entre a máquina LAB0623 e as outras. Ela se comportou dentro do padrão dos tempos das outras máquinas remotas.

O tempo de busca dos dados da máquina local comparada às máquinas remotas é bem menor. Isso pode ser explicado devido à existência da rede por onde os dados precisam trafegar até chegar à sua máquina destino e origem. Pode-se concluir, portanto, que quando se usa somente os recursos internos da máquina, o tempo de resposta é 1,25% mais rápido que usando um meio de comunicação externa (no caso a rede).

Entre as máquinas remotas os tempos são quase semelhantes, não havendo alterações bruscas, ou seja, tanto na última máquina (LAB0625) quanto na primeira máquina remota (LAB0622) os tempos não se alteraram muito.

Com relação à estrutura das tabelas não houve, também, alterações muito relevantes quanto às diferenças estruturais, ou seja, para a busca dos dados o tempo não aumenta com a estrutura complexa da tabela ou diminui com a simplicidade da mesma. Pode-se perceber também que a alteração do tempo nesse teste está sendo devido à quantidade de registros, pois, quanto mais registros para se recuperar, menos tempo o Oracle leva para retornar os dados.

Os dados deste teste podem ser visualizados nas Figuras 6.12, 6.13 e 6.14 a seguir.

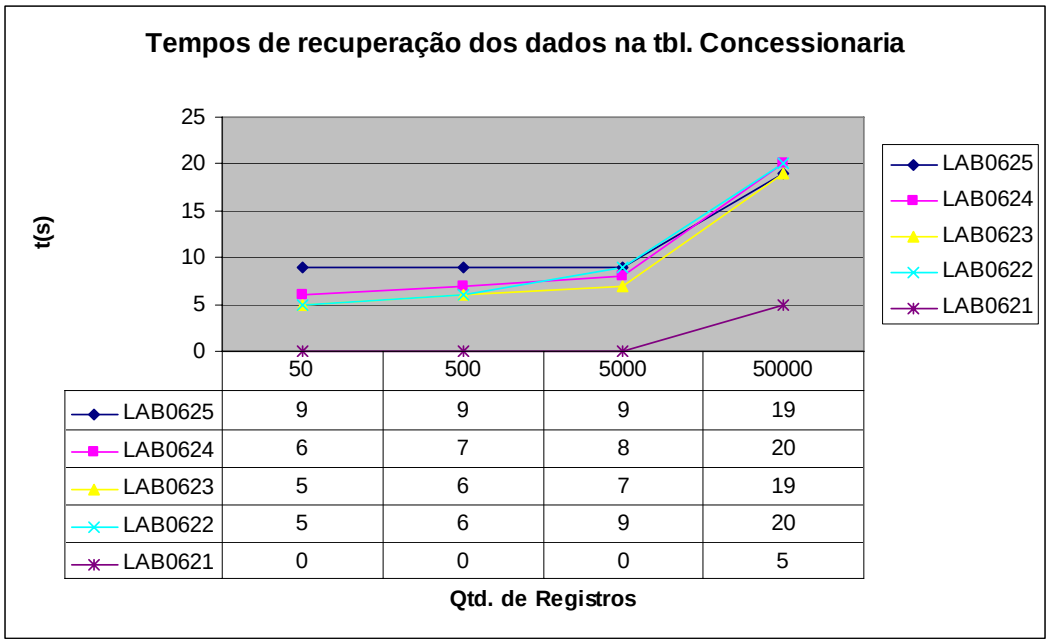


Figura 6.12: Comparação de recuperação dos dados na tabela Concessionária.

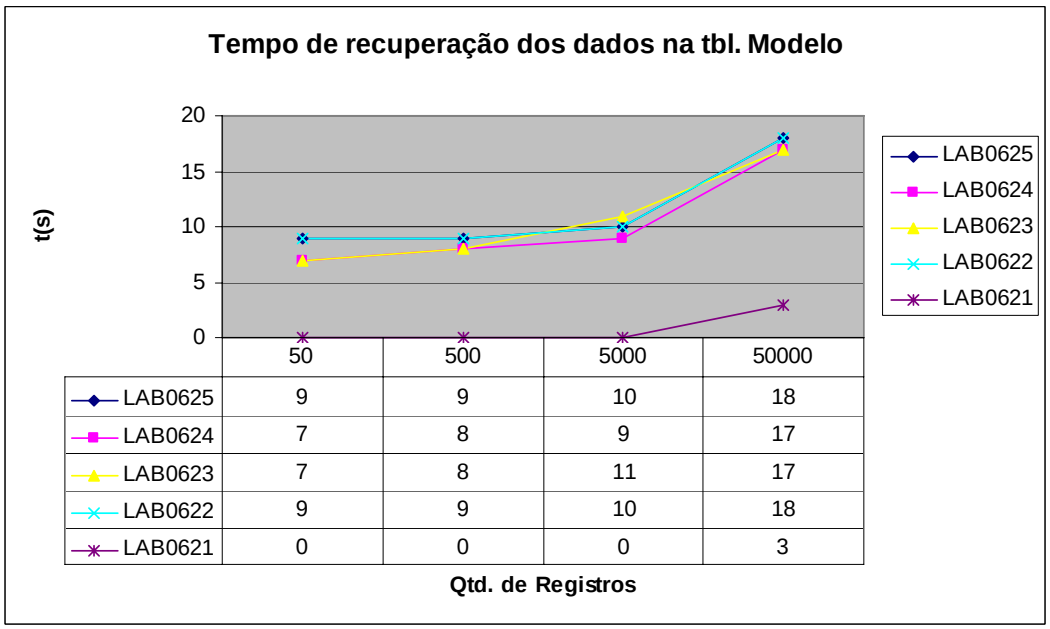
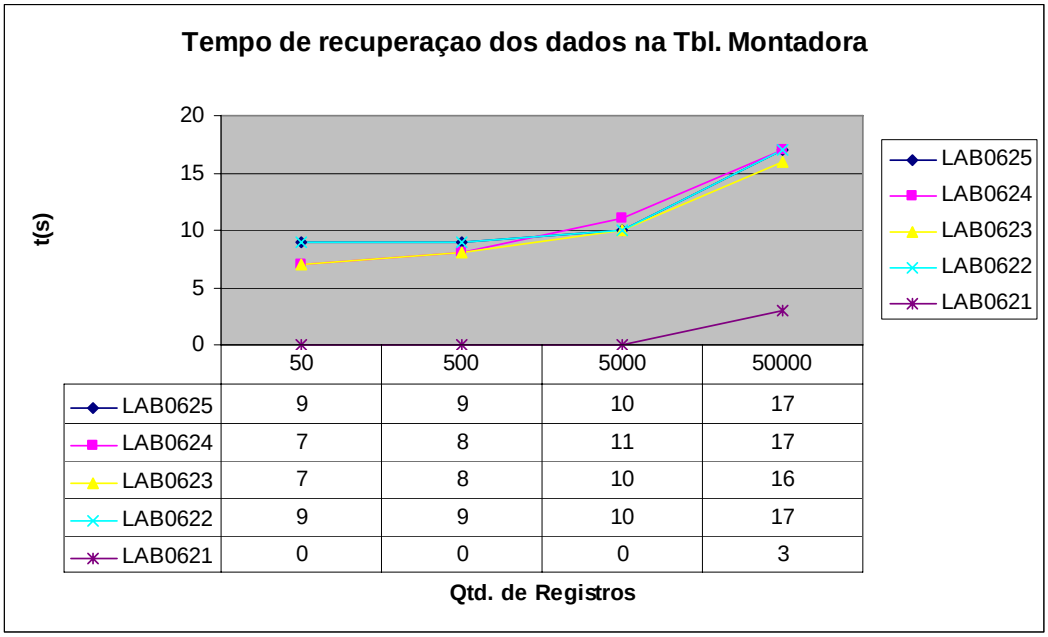


Figura 6.13: Comparação de recuperação dos dados na tabela Modelo.



	50	500	5000	50000
LAB0625	9	9	10	17
LAB0624	7	8	11	17
LAB0623	7	8	10	16
LAB0622	9	9	10	17
LAB0621	0	0	0	3

Figura 6.14: Comparação de recuperação dos dados na tabela Montadora.

6.3.3. Tabelas com testes diferenciados

Os testes executados nessa etapa foram feitos com as tabelas *Produto* e *Venda*, sendo que junto com a tabela *Produto* foi testado a tabela *Veículo*, *Peça* e *Opcionais*, e os testes da tabela *Venda* foi juntamente com a tabela *Itens_venda.*

Nesse teste, também, a máquina LAB0623 não demonstrou tempo de resposta maior que as outras tabelas remotas.

Para buscar os dados nessas tabelas foi preciso usar uma estrutura SQL diferente, pois os dados estão em diversas tabelas e não concentrados em uma só. Na Figura 6.15 são mostrados os comandos SQL usados para a recuperação dos dados na tabela Produto. O retorno que esse comando mostra é que todos os dados da tabela *Produto*, com os dados correspondentes na tabela *Veículo* e *Peça*, sendo que essa última tabela se junta aos dados

correspondentes à tabela *Opcionais*, ou seja, com esse comando pode-se visualizar se um produto é do tipo veículo ou peça, e se ele for do tipo peça pode-se ver se ele tem opcional.

```
SELECT * FROM PRODUTO
LEFT JOIN VEICULO ON
(PRODUTO.CODIGOBARRAS=VEICULO.CODIGOBARRAS)
LEFT JOIN PECA ON (PRODUTO.CODIGOBARRAS=PECA.CODIGOBARRAS)
LEFT JOIN OPCIONAIS ON
 (PRODUTO.CODIGOBARRAS =  OPCIONAIS.CODIGOBARRAS)
```

Figura 6.15: SQL usada para busca dos dados referentes a tabela Produto.

Nos testes de recuperação dos dados anteriores a esse os comandos SQL executados foram mais simples, não havendo a necessidade de juntar os dados de diversas tabelas em uma consulta.

A busca dos dados na tabela Venda também usou o comando LEFT JOIN como mostrada na Figura 6.16.

```
SELECT  * FROM VENDA
 LEFT JOIN ITENS_VENDA ON
(VENDA.NUMERO_DA_VENDA=ITENS_VENDA.NUMERO_DA_VENDA)
```

Figura 6.16: SQL usada para busca dos dados referentes a tabela Venda.

Em relação aos comandos SQL, a frase usada para recuperar dados da tabela *Produto* é um pouco mais complexa do que os comandos usados para consultar a tabela *Venda*, porém nos resultados demonstrados nas Figuras 6.17 e 6.18 os tempos de respostas quase não se alteraram, ou seja, na execução desta SQL, para o Oracle, a junção dos dados com mais duas ou três tabelas não exerceu influência relevante para alterar o tempo de processamento.

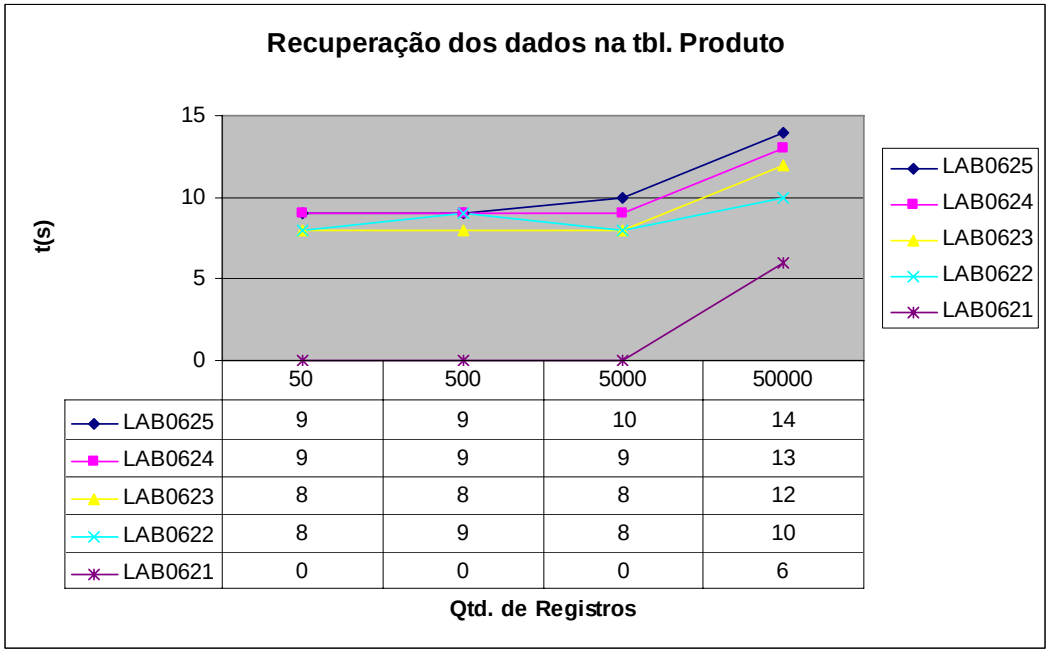


Figura 6.17: Comparação de recuperação dos dados na tabela Produto.

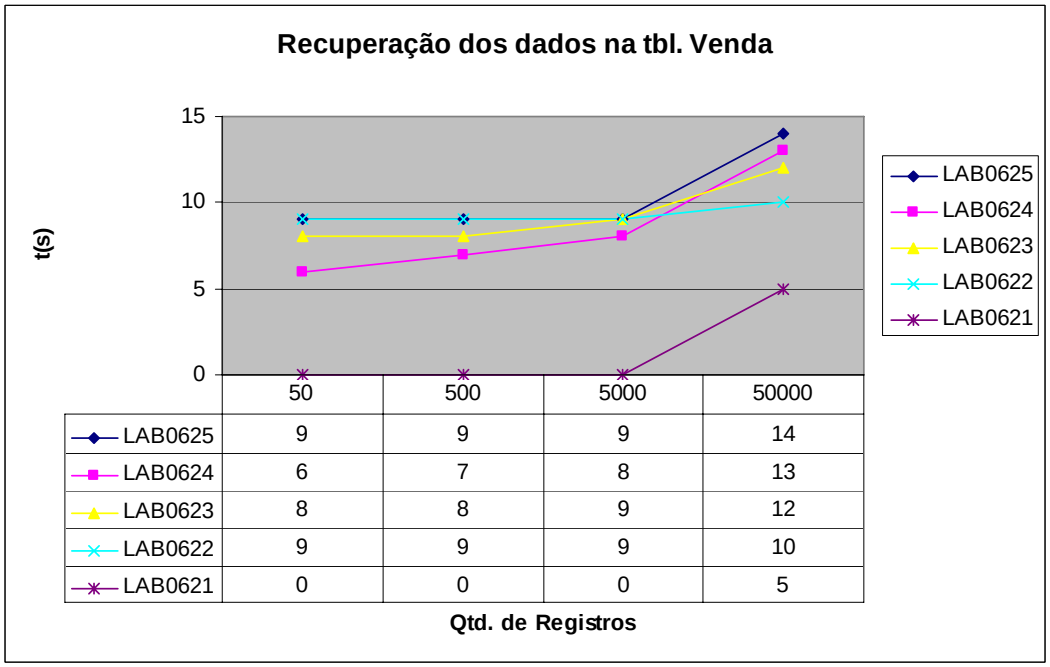


Figura 6.18: Comparação de recuperação dos dados na tabela Venda.

O tempo de resposta desses dois testes com base na quantidade de dados buscada mostrou uma diferença, pois o tempo de busca dos dados quando a quantidade é igual a 50

registros é maior do que a quantidade de 50.000 registros. Porém, no geral, não houve quase variação dos tempos comparando um teste com outro, mesmo as estruturas das tabelas sendo diferentes, ou seja, na busca dos dados a estrutura da tabela não altera o tempo.

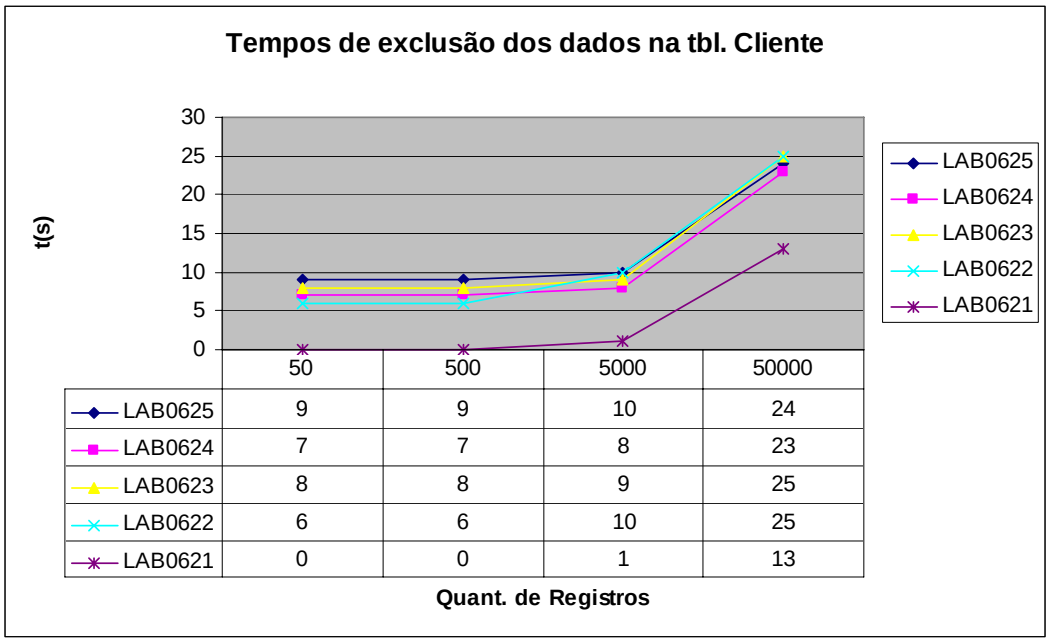
Comparando as tabelas locais (máquina LAB0621) com as remotas (as máquinas restantes) nos dois testes, pode-se verificar, novamente, que localmente o tempo de execução dos comandos é 9,20% mais rápido que o remoto.

6.4. Testes de Exclusão dos Dados

O teste de exclusão foi o último elaborado para esse trabalho. Não há grandes diferenças no código do programa para excluir os dados das tabelas, somente nas tabelas *Produto* e *Venda*, pois, como dito anteriormente elas estão sendo testadas juntamente com outras tabelas.

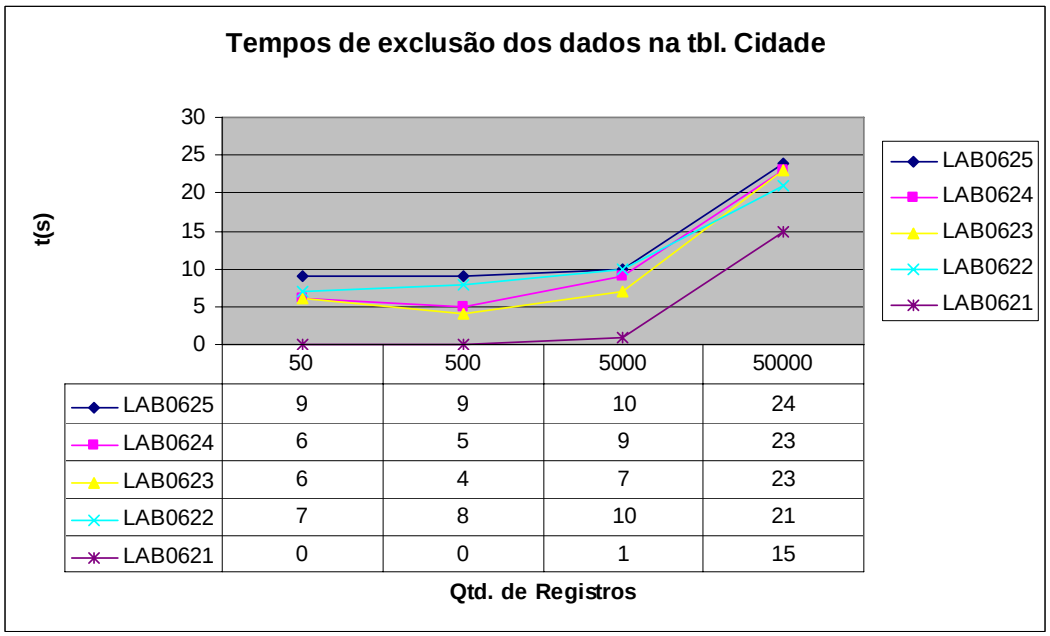
6.4.1. Tabelas com um Relacionamento

Os resultados dos testes das tabelas com um relacionamento podem ser vistos nas Figuras 6.19, 6.20 e 6.21 a seguir.



	50	500	5000	50000
LAB0625	9	9	10	24
LAB0624	7	7	8	23
LAB0623	8	8	9	25
LAB0622	6	6	10	25
LAB0621	0	0	1	13

Figura 6.19: Comparação de exclusão dos dados na tabela Cliente.



	50	500	5000	50000
LAB0625	9	9	10	24
LAB0624	6	5	9	23
LAB0623	6	4	7	23
LAB0622	7	8	10	21
LAB0621	0	0	1	15

Figura 6.20: Comparação de exclusão dos dados na tabela Cidade.

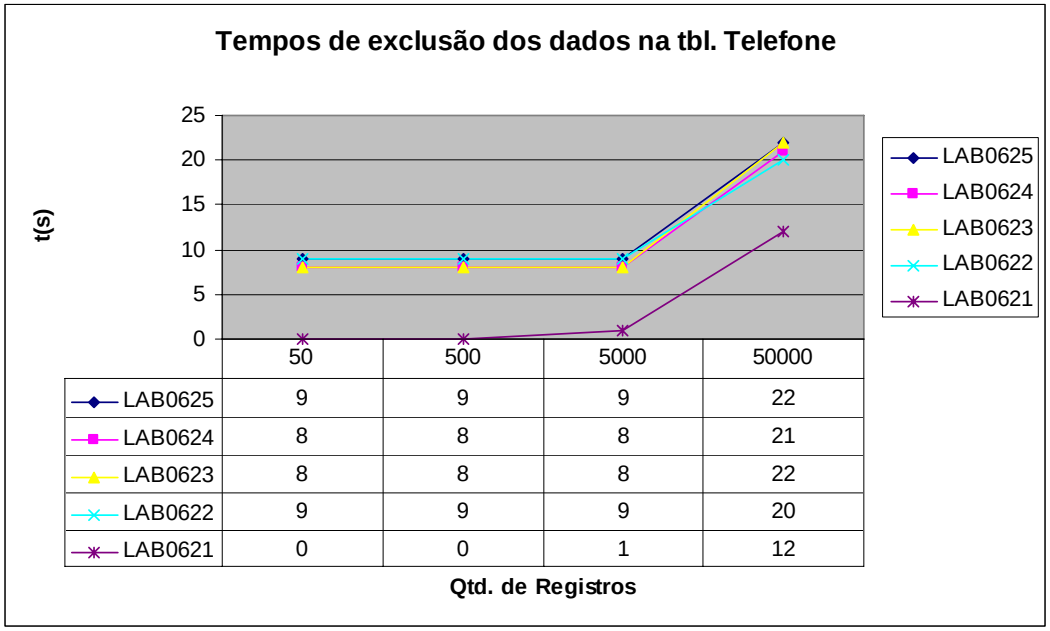


Figura 6.21: Comparação de exclusão dos dados na tabela Telefone.

Com base nas figuras anteriores e nas quantidades de registros igual a 50.00, pode-se perceber que o tempo da tabela *Cliente* é 3,77% maior que a tabela *Cidade* e a tabela *Cliente* em comparação à tabela *Telefone* é 13,40% maior. Em relação aos tempos das tabelas *Cidade* e *Telefone*, a tabela *Cidade* tem seu tempo 9,27% maior que a outra. Em relação a esta diferença, é possível que a estrutura da tabela *Cidade* tenha exercido influência, pois nessa tabela há dois campos do tipo *VARCHAR* enquanto a outra tabela possue mais campos com outros tipos de dados.

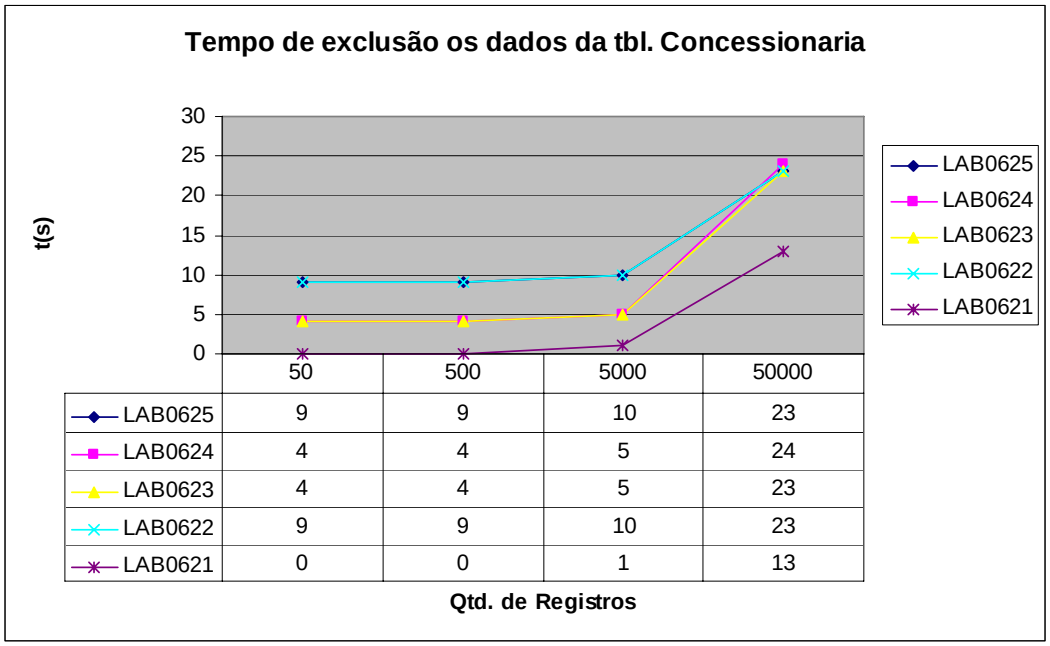
Em relação à máquina remota e a local, também há uma diferença significativa nos tempos, ou seja, mais uma vez comprova que localmente é mais rápido aproximadamente 11% que quando os dados estão remotos, devido à existência de mais recursos necessários para a execução da tarefa.

Na questão de quantidade de dados excluídos, o Oracle também retorna uma grande diferença no tempo entre pequenas e grandes quantidades, fazendo com que o tempo de processamento aumente conforme a quantidade de registros na tabela. Em comparação às

quantidades de 50 e 50.000 registros, pode-se verificar que a diferença dos tempos entre menores quantidades de registros é 452,33% maior que as quantidades maiores.

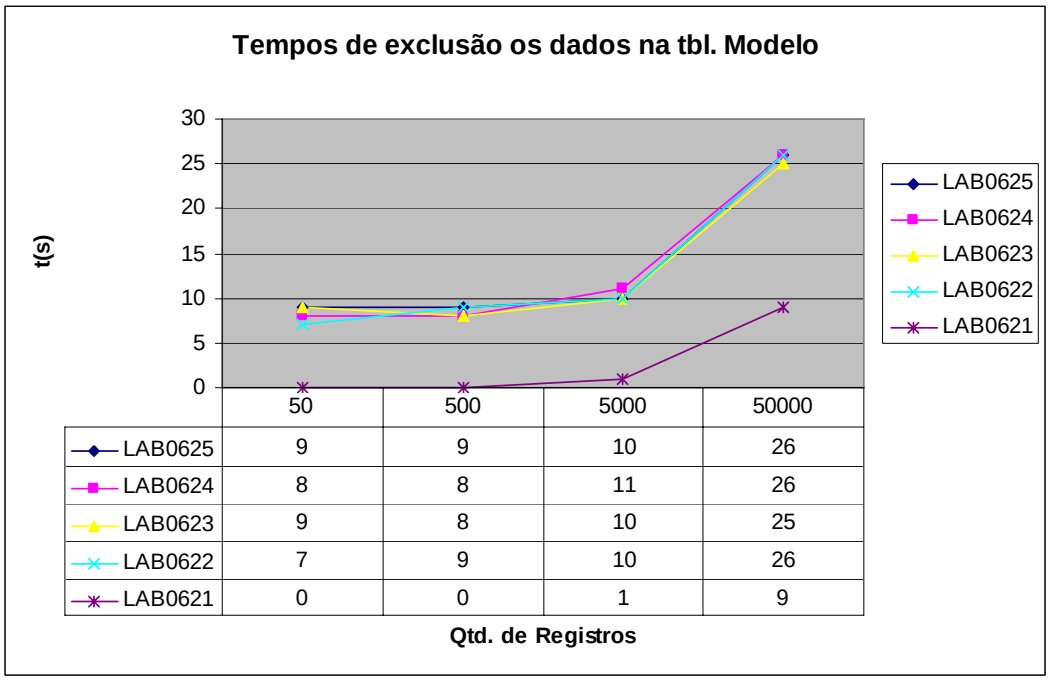
6.4.2. Tabelas com mais do que um Relacionamento

As tabelas referentes a este teste são: *Concessionária*, *Montadora* e *Modelo*. Com os testes pode-se notar que a quantidade de relacionamentos não interfere nos tempos da execução da exclusão dos dados, pois a tabela *Concessionária* é a que possui maior quantidade de relacionamento, porém possui um tempo um pouco menor até que as outras duas tabelas comparadas à quantidade de exclusão de 50.000 registros. Essas informações podem ser vistas nas Figuras 6.22, 6.23 e 6.24.



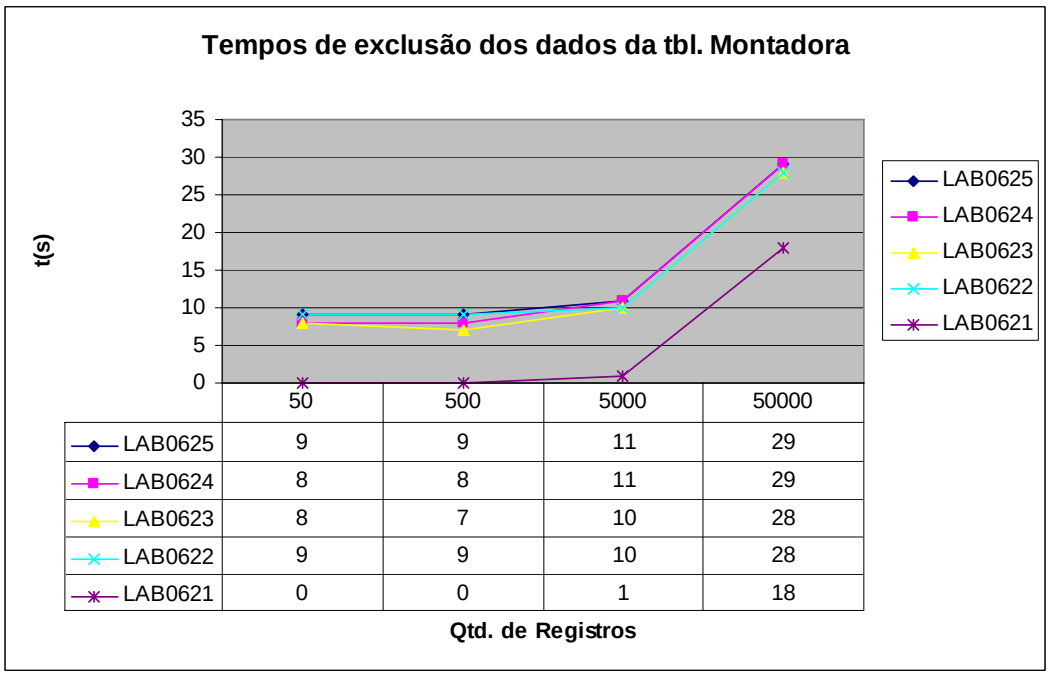
	50	500	5000	50000
LAB0625	9	9	10	23
LAB0624	4	4	5	24
LAB0623	4	4	5	23
LAB0622	9	9	10	23
LAB0621	0	0	1	13

Figura 6.22: Comparação de exclusão dos dados na tabela Concessionária.



	50	500	5000	50000
LAB0625	9	9	10	26
LAB0624	8	8	11	26
LAB0623	9	8	10	25
LAB0622	7	9	10	26
LAB0621	0	0	1	9

Figura 6.23: Comparação de exclusão dos dados na tabela Modelo.



	50	500	5000	50000
LAB0625	9	9	11	29
LAB0624	8	8	11	29
LAB0623	8	7	10	28
LAB0622	9	9	10	28
LAB0621	0	0	1	18

Figura 6.24: Comparação de exclusão dos dados na tabela Montadora.

Com relação à quantidade de registros pode-se ver que nas três tabelas testadas, quanto maior a quantidade de registros para excluir, menor o tempo gasto com a operação, chegando a uma média de 268,56% se comparado às quantidades com 50 e 50.000 registros. E que esse tempo, quando se refere às máquinas locais é bem menor do que nas máquinas remotas. Essa comparação de máquina local e remota seguiu sempre um mesmo padrão nos testes, sempre que é local o tempo é bem pequeno comparado com as outras máquinas remotas.

A comparação dos tempos com base na quantidade de registros também está seguindo um padrão dos testes anteriores: o tempo é maior quando há menos quantidade de registro e quando se tem mais quantidades de registros para excluir o tempo é menor.

6.4.3. Tabelas com testes diferenciados

Os testes executados agora foram feitos com as tabelas *Produto* e *Venda*, sendo que junto com a tabela *Produto* foi testado a tabela *Veículo*, *Peça* e *Opcionais*, e os testes da tabela *Venda* foram juntamente com a tabela *Itens_venda*.

Para a exclusão na tabela *Produto* foi preciso executar dois comandos. O primeiro na Figura 6.24, mostra que é preciso apagar todos os dados da tabela *Opcionais* antes de excluir os dados da tabela *Produto*. Porém não foi preciso apagar separadamente as tabelas *Veículo* e *Peça*, pois, na criação do modelo do banco de dados, foi criado um comando *DELETE CASCADE* para que quando um registro da tabela *Produto* fosse apagado imediatamente antes de apagar dessa tabela, ele apaga o registro correspondente, ou na tabela de *Veículo* ou *Peça,* e só em seguida apaga os dados na tabela *Produto.* Isso garante a integridade dos dados.

```
DELETE FROM OPCIONAIS;
DELETE FROM PRODUTO;
```

Figura 6.25: Comando usado no teste de exclusão na tabela Produto.

Na Figura 6.26 é demonstrado o código usado para excluir os dados dos testes da tabela *Venda*, onde ele não possuía na criação do banco o comando *DELETE CASCADE*, portanto foi preciso apagar os dados primeiramente da tabela filha (*Itens_Venda*) e depois a tabela de *Venda*.

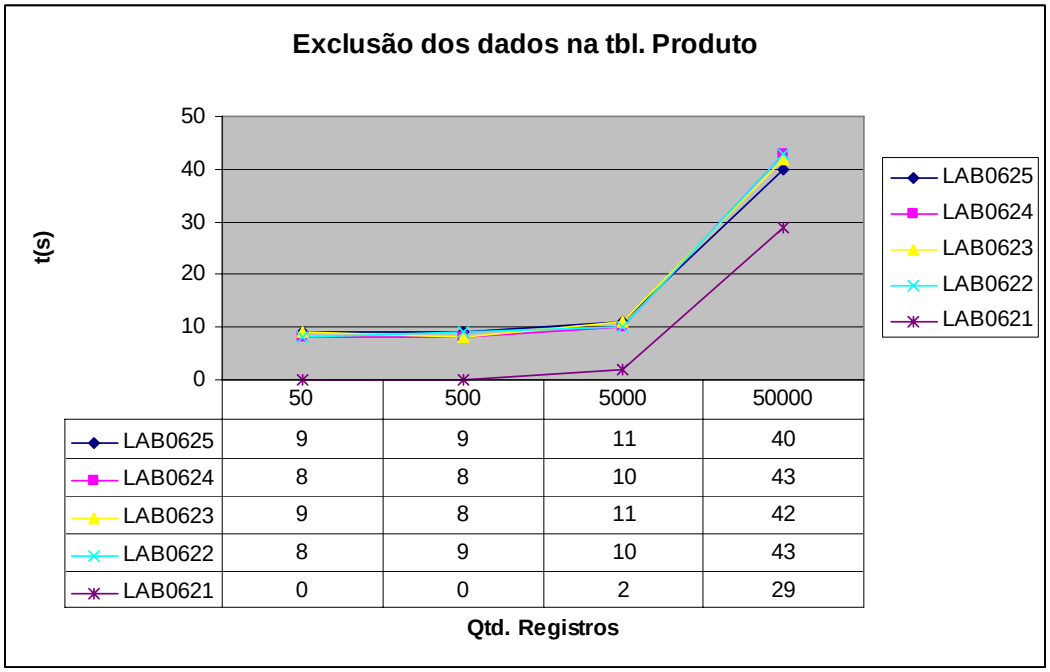
```
DELETE FROM ITENS_VENDA;
DELETE FROM VENDA;
```

Figura 6.26: Comando usado no teste de exclusão na tabela Venda.

Apesar das tabelas possuírem estruturas e relacionamentos diferentes é possível através das Figuras 6.27 e 6.28, estabelecer algumas diferenças nos tempos. Pode-se notar que os tempos do teste na tabela *Produto*, em relação à quantidade de 50.000, é 138,73% superior aos tempos da tabela *Venda* onde a diferença maior entre essas tabelas é o comando *DELETE CASCADE*, pois a exclusão da tabela *Venda*, como visto na Figura 6.26, é feita por dois comandos SQL separados. Podendo concluir por este teste, que o comando *DELETE CASCADE* aumenta o tempo de exclusão dos dados, porém a integridade do banco é mantida sem que nenhuma falha no programa deixe dados inconsistentes.

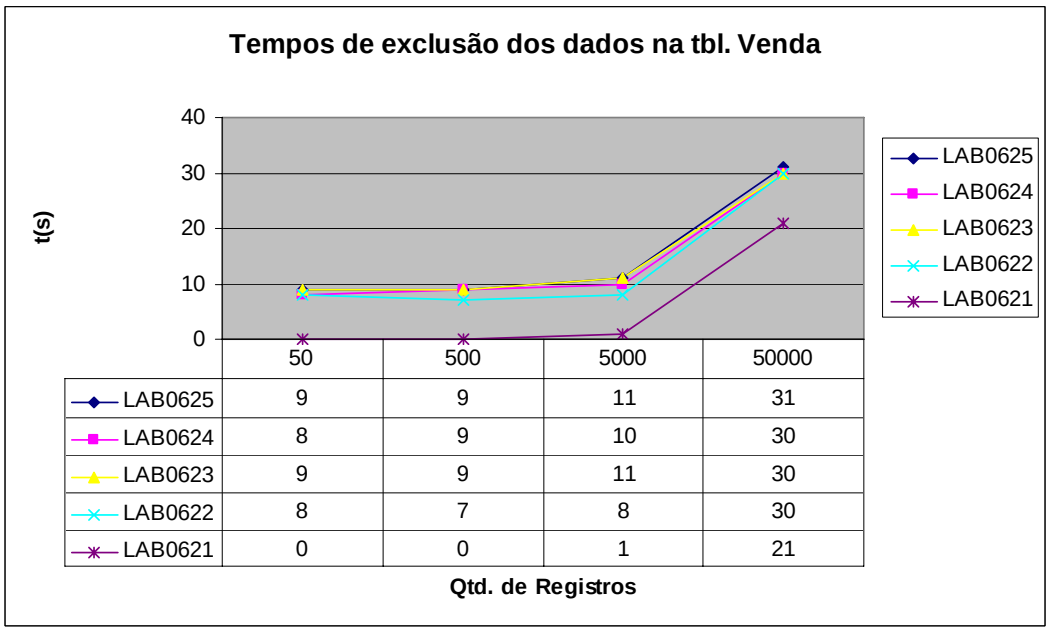
Novamente o comportamento observado sobre os tempos das máquinas remotas, locais e quantidade de registros se repetiu; assim, o tempo de execuções locais é 5,40% menor

que o tempo das máquinas remotas e, quanto mais registros para eliminar, menor o tempo que se leva para executar.



	50	500	5000	50000
LAB0625	9	9	11	40
LAB0624	8	8	10	43
LAB0623	9	8	11	42
LAB0622	8	9	10	43
LAB0621	0	0	2	29

Figura 6.27: Comparação de exclusão dos dados na tabela Produto.



	50	500	5000	50000
LAB0625	9	9	11	31
LAB0624	8	9	10	30
LAB0623	9	9	11	30
LAB0622	8	7	8	30
LAB0621	0	0	1	21

Figura 6.28: Comparação de exclusão dos dados na tabela Venda.

6.5. Considerações finais

Não houve explicação suficiente para se ter conclusão dos tempos elevados da máquina Lab0623 referente aos testes de inserção, porém foram pesquisadas informações sobre o HD da máquina em questão e os outros componentes da mesma, não havendo alterações relativas quanto aos componentes (placa mãe, memória e processador). A única diferença encontrada foi que a máquina Lab0623 possui 21,3 GB do HD usados e as demais possuem 19 GB usados.

7. CONCLUSÕES

A partir dos testes realizados neste trabalho foi possível avaliar o desempenho do SGBD Oracle9i quando executado de forma distribuída considerando dados convencionais (números e texto).

Verificou-se que este SGBD possibilita a configuração através de ferramentas gráficas, tornando-a mais amigável. Este oferece níveis de transparência, dando a ilusão ao usuário como um único SGBD local. O Oracle9i também fornece a capacidade de transações distribuídas, replicação e particionamento em locais múltiplos.

Com base nos testes apresentados anteriormente pode-se concluir que a operação de inserção requer mais tempo de processamento, seguido da operação de exclusão e por último a recuperação, que tem os menores tempos de execução.

Em relação aos relacionamentos, a tabela *Produto* tem o maior tempo nas operações de inserção e exclusão, ou seja, tabelas com especializações de dados, *DELETE CASCADE* e *CHECK* aumentam o tempo de processamento, porém garantem a integridade dos dados.

O segundo maior tempo é dos testes com a tabela *Venda* onde essa exigiu que existissem dados em todas as outras tabelas do modelo do banco de dados tornando-a assim mais complexa que as outras tabelas, como a *Cidade*, *Telefone*, *Modelo*, *Montadora* e *Concessionária*. Portanto, a velocidade de processamento em relação aos testes de inserção e exclusão depende da complexidade do relacionamento entre as tabelas.

Em relação à recuperação de dados, observou-se praticamente os mesmos tempos em todas as tabelas, o que leva a afirmar que não há grandes influências de relacionamentos e estruturas.

O restante das tabelas apresentou tempos menores que a tabela *Produto* e *Venda*, porém, os tempos também estão diretamente relacionados à complexidade dos relacionamentos entre elas.

REFERÊNCIAS BIBLIOGRÁFICAS

Apostila Java. Disponível em: http://www.apostilando.com/download_final.php?cod=35. Acesso em: 10 nov. 2005.

ARANTES,Adriano Siqueira; FIGUEIREDO, Josiel Maimone; BIAJIZ, Mauro. **Replicação**, 2005.

ARANTES,Adriano Siqueira; FIGUEIREDO, Josiel Maimone; BIAJIZ, Mauro. **Introdução a Banco de Dados Distribuídos**, 2005.

ARANTES,Adriano Siqueira; FIGUEIREDO, Josiel Maimone; BIAJIZ, Mauro. **Processamento de Consultas distribuídas**, 2005.

ARANTES,Adriano Siqueira; FIGUEIREDO, Josiel Maimone; BIAJIZ, Mauro. **Gerenciamento de Transações Distribuídas**, 2005.

Banco de Dados Relacional. Disponível em: http://pt.wikipedia.org/wiki/Banco_de_dados_relacional. Acessado em: 13 jun. 2006.

CERÍCOLA, V. Oswald. **Oracle Banco de Dados Relacional e Distribuído,** Makron Books, 1995.

CESTA, André Augusto. **A Linguagem de Programação JAVA**, 2004. Disponível em: http://www.apostilando.com/download.php?cod=37&categoria=Java. Acesso em: 13 junho 2005.

DATE, C. J.**Introdução a Sistemas de Bancos de Dados**, Campus, 1990.

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

HOLM B., CARNELL J., GOODMAM J., MARCOTTE B., MURKHAR K., NARANJO M., PIERMARINI M., RAAJ A., SARANG Dr. P., SINGH S., STUBBS T. **Oracle 9i Java Programando**, Alta Books, 2002.

JANDL, Peter. **Introdução ao Java**, 1999. Disponível em: http://www.apostilando.com/download.php?cod=36&categoria=Java. Acesso em: 25 agosto 2005.

LONEY, KEVIN, THERIAULT, MARLENE. **Oracle 9i O manual do DBA,** Campus, 2002.

NAVATHE, SHAMKANT B., ELMASRI, RAMEZ. **Sistemas de Banco de Dados,** Pearson, 2005.

Oracle Corporation. Disponível em**:** http://www.oracle.com**.** Acessado em: 06 abril 2006.

Oracle 9i DBA Fundamentals II. Student Guide – Volume 1, 2005.
Installing Oracle 9i.

OZSU, M. Tamer; VALDURIEZ, Patrick. **Princípios de Sistemas de Bancos de Dados Distribuídos**, CAMPUS, 2001.

RODRIGUES, C. J. O., **Monografia: "Armazenamento e Recuperação de Imagens Médicas Utilizando o SGBD Oracle9i Distribuído.",** Marília – SP: Centro Universitário Eurípides de Marilia, 2005.

SILBERSCHATZ, A., KORTH, H.F, SUDARSHAN, S., **Sistemas de Banco de Dados**,Makron Books, 1999.

SILVA, Enio Kilder Oliveira**. Monografia: "Um Estudo sobre Sistemas de Banco de Dados Cliente/Servidor."** João Pessoa - PB: Faculdade Paraibana de Processamento de Dados, 2001.

SURIAN, Jorge, NICOCHELLI, Luiz. **Apostila de Banco de Dados e SQL**, 1999. Disponível em: http://portal.tol.pro.br/portal/index.php?option=com_remository&func=fileinfo&parent=category&filecatid=47. Acesso em: 25 mar. 2005.

VALERO, Nuno Ribeiro. **Sistemas Distribuídos, 2005. Disponível em:** ltodi.est.ips.pt/nribeiro/Lecturing/SD_02-03/Cap1.ppt. Acessado em: 20 jun. 2005.

VELLOSO, Fernando de Castro. **Informática Conceitos Básicos**, CAMPUS, 1997.

ANEXO A – *Script* gerado a partir do modelo do banco de dados criado.

```
CREATE TABLE Cidade (
    UF              VARCHAR2(03) NULL
                      CHECK (UF IN ('SP', 'MG', 'RJ', 'ES', 'PR', 'SC', 'RS')),
    Nome            VARCHAR2(60) NULL,
    Codigo          NUMBER NOT NULL
);


ALTER TABLE Cidade
    ADD PRIMARY KEY (Codigo);


CREATE TABLE Cliente (
    Endereco        VARCHAR2(40) NULL,
    Telefone        VARCHAR2(20) NOT NULL,
    Nome            VARCHAR2(80) NOT NULL,
    CPF             NUMBER NOT NULL
);


ALTER TABLE Cliente
    ADD PRIMARY KEY (CPF);


CREATE TABLE Concessionaria (
    Complemento       VARCHAR2(20) NULL,
    Bairro          VARCHAR2(20) NULL,
    CEP             NUMBER NULL,
    Numero          NUMBER NULL,
    Logradouro      VARCHAR2(40) NULL,
    Nome            VARCHAR2(80) NULL,
    Codigo          NUMBER NOT NULL,
    Codigo_da_Cidade    NUMBER NULL,
    Codigo_da_Montadora  NUMBER NOT NULL,
    CNPJ            NUMBER NOT NULL
);


ALTER TABLE Concessionaria
    ADD PRIMARY KEY (Codigo, Codigo_da_Montadora);
```

```
CREATE TABLE Itens_Venda (
    Numero_da_Venda     NUMBER NOT NULL,
    CodigoBarras        VARCHAR2(20) NOT NULL,
    Quantidade          NUMBER NOT NULL
);


ALTER TABLE Itens_Venda
    ADD PRIMARY KEY (Numero_da_Venda, CodigoBarras);


CREATE TABLE Modelo (
    NumeroPortas        NUMBER NULL,
    Tipo                VARCHAR2(02) NULL
                          CHECK (Tipo IN ('S', 'H', 'P')),
    Combustivel         VARCHAR2(10) NULL,
    Motor               VARCHAR2(10) NULL,
    Ano                 NUMBER NULL,
    Codigo              NUMBER NOT NULL,
    Codigo_da_Montadora  NUMBER NOT NULL
);


ALTER TABLE Modelo
    ADD PRIMARY KEY (Codigo, Codigo_da_Montadora);


CREATE TABLE Montadora (
    Contato             VARCHAR2(20) NULL,
    Endereco            VARCHAR2(40) NULL,
    RazaoSocial         VARCHAR2(80) NOT NULL,
    Nome_Fantasia       VARCHAR2(80) NOT NULL,
    CNPJ                NUMBER NOT NULL,
    Codigo              NUMBER NOT NULL
);


ALTER TABLE Montadora
    ADD PRIMARY KEY (Codigo);


CREATE TABLE Opcionais (
    Codigo              NUMBER NOT NULL,
    Codigo_da_Montadora  NUMBER NOT NULL,
    CodigoBarras        VARCHAR2(30) NOT NULL );
```

```
ALTER TABLE Opcionais
     ADD PRIMARY KEY (Codigo, Codigo_da_Montadora, CodigoBarras);


CREATE TABLE Peca (
     Peso              NUMBER NULL,
     Numero_de_Serie     NUMBER NULL,
     CodigoBarras        VARCHAR2(20) NOT NULL
);


ALTER TABLE Peca
     ADD PRIMARY KEY (CodigoBarras);


CREATE TABLE Produto (
     Valor_Venda         DECIMAL(19,4) NULL,
     Valor_Unitario      DECIMAL(19,4) NULL,
     Descricao          VARCHAR2(60) NULL,
     CodigoBarras        VARCHAR2(20) NOT NULL,
     Tipo_Produto        SMALLINT NULL
                        CHECK (Tipo_Produto IN ('V', 'P'))
);


ALTER TABLE Produto
     ADD PRIMARY KEY (CodigoBarras);


CREATE TABLE Telefone (
     Codigo_da_Concessionaria NUMBER NOT NULL,
     Codigo_da_Montadora  NUMBER NOT NULL,
     Numero_Telefone      NUMBER NOT NULL,
     Tipo_do_Telefone     VARCHAR2(02) NULL
                        CHECK (Tipo_do_Telefone IN ('C', 'F', 'T', 'X', 'R'))
);


ALTER TABLE Telefone
     ADD PRIMARY KEY (Codigo_da_Concessionaria, Codigo_da_Montadora,
          Numero_Telefone);
```

```
CREATE TABLE Veiculo (
     Avaliacao_Teste_Seguranca NUMBER NULL,
     Numero_do_Chassis    VARCHAR2(20) NULL,
     Cor                  VARCHAR2(20) NULL,
     CodigoBarras         VARCHAR2(20) NOT NULL,
     Codigo_do_Modelo     NUMBER NOT NULL,
     Codigo_da_Montadora  NUMBER NOT NULL
);


ALTER TABLE Veiculo
     ADD PRIMARY KEY (CodigoBarras);


CREATE TABLE Venda (
     Data               DATE NULL,
     Numero_da_Venda      NUMBER NOT NULL,
     Codigo_da_Concessionaria NUMBER NOT NULL,
     Codigo_da_Montadora  NUMBER NOT NULL,
     CPF_do_Cliente       NUMBER NOT NULL
);


ALTER TABLE Venda
     ADD PRIMARY KEY (Numero_da_Venda);


ALTER TABLE Concessionaria
     ADD FOREIGN KEY (Codigo_da_Montadora)
                    REFERENCES Montadora  (Codigo);
ALTER TABLE Concessionaria
     ADD FOREIGN KEY (Codigo_da_Cidade)
                    REFERENCES Cidade  (Codigo)
                    ON DELETE SET NULL;


ALTER TABLE Itens_Venda
     ADD FOREIGN KEY (CodigoBarras)
                    REFERENCES Produto  (CodigoBarras);

ALTER TABLE Itens_Venda
     ADD FOREIGN KEY (Numero_da_Venda)
                    REFERENCES Venda  (Numero_da_Venda);
```

```
ALTER TABLE Modelo
    ADD FOREIGN KEY (Codigo_da_Montadora)
                REFERENCES Montadora  (Codigo);


ALTER TABLE Opcionais
    ADD FOREIGN KEY (CodigoBarras)
                REFERENCES Peca  (CodigoBarras);


ALTER TABLE Opcionais
    ADD FOREIGN KEY (Codigo, Codigo_da_Montadora)
                REFERENCES Modelo  (Codigo,
        Codigo_da_Montadora);


ALTER TABLE Peca
    ADD FOREIGN KEY (CodigoBarras)
                REFERENCES Produto  (CodigoBarras)
                ON DELETE CASCADE;


ALTER TABLE Telefone
    ADD FOREIGN KEY (Codigo_da_Concessionaria, Codigo_da_Montadora)
                REFERENCES Concessionaria  (Codigo,
        Codigo_da_Montadora);


ALTER TABLE Veiculo
    ADD FOREIGN KEY (Codigo_do_Modelo, Codigo_da_Montadora)
                REFERENCES Modelo  (Codigo,
        Codigo_da_Montadora);


ALTER TABLE Veiculo
    ADD FOREIGN KEY (CodigoBarras)
                REFERENCES Produto  (CodigoBarras)
                ON DELETE CASCADE;


ALTER TABLE Venda
    ADD FOREIGN KEY (CPF_do_Cliente)
                REFERENCES Cliente  (CPF);
```

```
ALTER TABLE Venda
    ADD FOREIGN KEY (Codigo_da_Concessionaria, Codigo_da_Montadora)
                REFERENCES Concessionaria  (Codigo,
        Codigo_da_Montadora);
```

ANEXO B – Programa para inserir, recuperar e excluir os dados.

```
package tcc;
import java.sql.*;
import oracle.jdbc.util.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import javax.swing.JOptionPane;
import java.sql.SQLException;
import java.io.*; //para o TXT
import java.text.DecimalFormat;

public class FrmPrincipal extends javax.swing.JFrame {

   public Connection connection = null;

       public FrmPrincipal() {

       initComponents();

    }

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
     jTextArea1.setText("");
     String QtReg, VOpcao;
     QtReg= jTextField1.getText();
     double v=0;
     try
       {
          String driverName = "oracle.jdbc.driver.OracleDriver";
          Class.forName(driverName);
       }
          catch (Exception e)
          {
              jTextArea1.append("Não pode carregar o drive: "+e);
          }

       try
       {
          // Create a connection to the database
          String serverName = "lab0621";
          String portNumber = "1521";
          String sid = "TCCBD21";
          String url = "jdbc:oracle:thin:@" + serverName + ":" + portNumber + ":" + sid;
          String username = "SYSTEM";
          String password = "vr1326";
```

```
connection = DriverManager.getConnection(url, username, password);
        }
        catch (Exception e)
           {
           jTextArea1.append("Não pode conectar: "+e);
           }

         // opção escolhida na tela
           VOpcao="";
           if (jRadioButton1.isSelected()) //insert
           {
              VOpcao="insert";
           }
            else if (jRadioButton2.isSelected()) //select
           {
             VOpcao= "select";
           }
           else if (jRadioButton4.isSelected()) //delete
           {
              VOpcao= "delete";
           }
           //fim da opção escolhida

        //execução dos comandos.....
        if (VOpcao== "insert")
        {
         try
           {
           jTextArea1.append("O numero de Registro para inserção é: "+QtReg+"\n");
           //recebe um statement de conexao
           Statement stmt= connection.createStatement();
           long timestamp=System.currentTimeMillis();
           double soma;
           float time2;
           String tipo = "V";
           QtReg= jTextField1.getText();
           v=1;
           soma=v+Integer.parseInt(QtReg); //transformar string em integer.
           int SomaCount = 0;
           long inicio=System.currentTimeMillis();
            while ((v+1) <= soma)
```

```
        int res= stmt.executeUpdate(VOpcao+" into  teste55 values (SysDate, "+v+", 1,1,1
)");
              res= stmt.executeUpdate(VOpcao+" into  teste60 values ("+v+", '1',02 )");
              v+=1;
             }
           long fim=System.currentTimeMillis();
           long t= fim - inicio;
           time2= t/1000; //para achar em segundos o tempo
          //  time=System.currentTimeMillis()-timestamp;
           //    time2= time/1000; //para achar em segundos o tempo.
           jTextArea1.append("\nO tempo gasto para Inserção do(s) dado(s) na maquina
Lab0625 foi: "+time2+" segundos\n");
           connection.commit();
           //outra tbl 24
           QtReg= jTextField1.getText();
           soma=v+Integer.parseInt(QtReg); //transformar string em integer.
           SomaCount = 0;
           tipo = "P";
           inicio=System.currentTimeMillis();
           while ((v+1) <= soma)
             {
                int res= stmt.executeUpdate(VOpcao+" into  teste54 values (SysDate, "+v+",
1,1,1 )");
                res= stmt.executeUpdate(VOpcao+" into  teste59 values ("+v+", '1',02 )");
               v+=1;
             }
           fim=System.currentTimeMillis();
           t= fim - inicio;
           time2= t/1000; //para achar em segundos o tempo
           jTextArea1.append("\nO tempo gasto para Inserção do(s) dado(s) na maquina
lab0624 foi: "+time2+" segundos\n");
           connection.commit();
           //
           QtReg= jTextField1.getText();
           soma=v+Integer.parseInt(QtReg); //transformar string em integer.
           SomaCount = 0;
           tipo = "V";
            while ((v+1) <= soma)
             {   int res= stmt.executeUpdate(VOpcao+" into  teste53 values (SysDate, "+v+",
1,1,1 )");
               res= stmt.executeUpdate(VOpcao+" into  teste58 values ("+v+", '1',02 )");
                 v+=1;
             }
```

```
 fim=System.currentTimeMillis();
          t= fim - inicio;
          time2= t/1000; //para achar em segundos o tempo
         // time=System.currentTimeMillis()-timestamp;
          //    time2= time/1000; //para achar em segundos o tempo.
          jTextArea1.append("\nO tempo gasto para Inserção do(s) dado(s) na maquina
lab0623 foi: "+time2+" segundos\n");
          connection.commit();
          //
          //outra tbl 22
          QtReg= jTextField1.getText();
          soma=v+Integer.parseInt(QtReg); //transformar string em integer.
          SomaCount = 0;
          tipo = "P";
         // v=0;
          inicio=System.currentTimeMillis();
          while ((v+1) <= soma)
            {     int res= stmt.executeUpdate(VOpcao+" teste52 values (SysDate, "+v+",
1,1,1 )");
             res= stmt.executeUpdate(VOpcao+"  teste57 values ("+v+", '1',02 )");
             v+=1;
            }
          fim=System.currentTimeMillis();
          t= fim - inicio;
          time2= t/1000; //para achar em segundos o tempo
          jTextArea1.append("\nO tempo gasto para Inserção do(s) dado(s) na maquina
Lab0622 foi: "+time2+" segundos\n");
          connection.commit();
          //
          //outra tbl 21
          QtReg= jTextField1.getText();
          soma=v+Integer.parseInt(QtReg); //transformar string em integer.
          SomaCount = 0;
          tipo = "V";
         // v=0;
          inicio=System.currentTimeMillis();
          while ((v+1) <= soma)
          {
            int res= stmt.executeUpdate(VOpcao+" into  teste51 values (SysDate, "+v+",
1,1,1 )");
            res= stmt.executeUpdate(VOpcao+" into  teste56 values ("+v+", '1',02 )");
             v+=1;
          }
```

```
fim=System.currentTimeMillis();
          t= fim - inicio;
          time2= t/1000; //para achar em segundos o tempo
          jTextArea1.append("\nO tempo gasto para Inserção do(s) dado(s) na maquina
lab0621 foi: "+time2+" segundos\n");
          connection.commit();
          //
          stmt.close();
          jTextArea1.append("\nRegistros criados com sucesso na tabela");
          //salva no TXT
          File outputFile = new File("Resposta.txt");
          FileWriter out = new FileWriter(outputFile);
          out.write(""+ jTextArea1.getText() + "");
          out.close();
          //fim do TXT
       }
          catch (Exception e)
          {
          //retorna o erro
          JOptionPane.showMessageDialog(null,"Impossível Inserir Dados, pelo Seguinte
Motivo:\n"+e,"Atenção",JOptionPane.WARNING_MESSAGE);
          }
          finally
          {

          }
       }  //fim da VOpcao insert
    if (VOpcao== "select")
        {
         try
           {
          jTextArea1.append("O numero de Registro para seleção é: "+QtReg+"\n");
          //recebe um statement de conexao
          Statement stmt= connection.createStatement();
          long timestamp=System.currentTimeMillis();
          double time,time2, soma;
          //salva no TXT
          File outputFile = new File("Resposta.txt");
          FileWriter out = new FileWriter(outputFile);
          //txt
          long inicio=System.currentTimeMillis();
          ResultSet rs= stmt.executeQuery(VOpcao+" * from teste55 LEFT JOIN teste ON
(teste55.NUMERO_DA_VENDA= teste60.NUMERO_DA_VENDA)  ");
```

```
while (rs.next()){
//VENDAS
            out.write(""+
"NUMERO_DA_VENDA:"+rs.getDouble("NUMERO_DA_VENDA")+"\n" + "");
            out.write(""+ "DATA:"+rs.getDate("DATA")+"\n" + "");
            out.write(""+
"CODIGO_DA_CONCESSIONARIA:"+rs.getDouble("CODIGO_DA_CONCESSIONA
RIA")+"\n" + "");
            out.write(""+
"CODIGO_DA_MONTADORA:"+rs.getDouble("CODIGO_DA_MONTADORA")+"\n
" + "");
            out.write(""+ "
CPF_DO_CLIENTE:"+rs.getDouble("CPF_DO_CLIENTE")+"\n" + "");
            out.write(""+ "CODIGOBARRAS:"+rs.getString("CODIGOBARRAS")+"\n" +
"");
            out.write(""+ "QUANTIDADE:"+rs.getDouble("QUANTIDADE")+"\n" + "");
           //FIM VENDAS
           }
           long fim=System.currentTimeMillis();
           rs.close();
           jTextArea1.append("Tempo inicio:"+inicio+"\n");
           jTextArea1.append("Tempo final:"+fim+"\n");
           long t= fim - inicio;
           DecimalFormat decimal = new DecimalFormat( "0.00" );
           time2= t/1000; //para achar em segundos o tempo
           decimal.format(time2);
           jTextArea1.append("\nO tempo gasto para Recuperação do(s) dado(s) na maquina
Lab0625 foi: "+time2+" segundos\n");
           jTextArea1.append("\nRegistros selecionados com sucesso na tabela");
          jTextArea1.append("O numero de Registro para seleção é: "+QtReg+"\n");
           //recebe um statement de conexao
           inicio=System.currentTimeMillis();
           rs= stmt.executeQuery(VOpcao+" * from teste54 LEFT JOIN teste59 ON
(teste54.NUMERO_DA_VENDA=teste59.NUMERO_DA_VENDA)  ");
            while (rs.next()){
//VENDAS
            out.write(""+
"NUMERO_DA_VENDA:"+rs.getDouble("NUMERO_DA_VENDA")+"\n" + "");
            out.write(""+ "DATA:"+rs.getDate("DATA")+"\n" + "");
            out.write(""+
"CODIGO_DA_CONCESSIONARIA:"+rs.getDouble("CODIGO_DA_CONCESSIONA
RIA")+"\n" + "");
```

```
out.write(""+
"CODIGO_DA_MONTADORA:"+rs.getDouble("CODIGO_DA_MONTADORA")+"\n
" + "");
            out.write(""+
"CPF_DO_CLIENTE:"+rs.getDouble("CPF_DO_CLIENTE")+"\n" + "");
            out.write(""+ "CODIGOBARRAS:"+rs.getString("CODIGOBARRAS")+"\n" +
"");
            out.write(""+ "QUANTIDADE:"+rs.getDouble("QUANTIDADE")+"\n" + "");
           //FIM VENDAS
           }
           fim=System.currentTimeMillis();
           rs.close();
           jTextArea1.append("Tempo inicio:"+inicio+"\n");
           jTextArea1.append("Tempo final:"+fim+"\n");
           t= fim - inicio;
           decimal = new DecimalFormat( "0.00" );
           time2= t/1000; //para achar em segundos o tempo
           decimal.format(time2);
           jTextArea1.append("\nO tempo gasto para Recuperação do(s) dado(s) na maquina
Lab0624 foi: "+time2+" segundos\n");
//cliente@tccbd23
            jTextArea1.append("O numero de Registro para seleção é: "+QtReg+"\n");
           //recebe um statement de conexao
           inicio=System.currentTimeMillis();
           rs= stmt.executeQuery(VOpcao+" * from teste53 LEFT JOIN teste58 ON
(teste53.NUMERO_DA_VENDA=teste58.NUMERO_DA_VENDA)  ");
           while (rs.next()){
//VENDAS
            out.write(""+
"NUMERO_DA_VENDA:"+rs.getDouble("NUMERO_DA_VENDA")+"\n" + "");
            out.write(""+ "DATA:"+rs.getDate("DATA")+"\n" + "");
            out.write(""+
"CODIGO_DA_CONCESSIONARIA:"+rs.getDouble("CODIGO_DA_CONCESSIONA
RIA")+"\n" + "");
            out.write(""+
"CODIGO_DA_MONTADORA:"+rs.getDouble("CODIGO_DA_MONTADORA")+"\n
" + "");
            out.write(""+ "
CPF_DO_CLIENTE:"+rs.getDouble("CPF_DO_CLIENTE")+"\n" + "");
            out.write(""+ "CODIGOBARRAS:"+rs.getString("CODIGOBARRAS")+"\n" +
"");
            out.write(""+ "QUANTIDADE:"+rs.getDouble("QUANTIDADE")+"\n" + "");
           //FIM VENDAS
           }
```

```
out.write(""+
"CODIGO_DA_MONTADORA:"+rs.getDouble("CODIGO_DA_MONTADORA")+"\n
" + "");
            out.write(""+
"CPF_DO_CLIENTE:"+rs.getDouble("CPF_DO_CLIENTE")+"\n" + "");
            out.write(""+ "CODIGOBARRAS:"+rs.getString("CODIGOBARRAS")+"\n" +
"");
            out.write(""+ "QUANTIDADE:"+rs.getDouble("QUANTIDADE")+"\n" + "");
           //FIM VENDAS
          }
          fim=System.currentTimeMillis();
          rs.close();
          jTextArea1.append("Tempo inicio:"+inicio+"\n");
          jTextArea1.append("Tempo final:"+fim+"\n");
          t= fim - inicio;
          decimal = new DecimalFormat( "0.00" );
          time2= t/1000; //para achar em segundos o tempo
          decimal.format(time2);
          jTextArea1.append("\nO tempo gasto para Recuperação do(s) dado(s) na maquina
Lab0624 foi: "+time2+" segundos\n");
//cliente@tccbd23
           jTextArea1.append("O numero de Registro para seleção é: "+QtReg+"\n");
          //recebe um statement de conexao
          inicio=System.currentTimeMillis();
          rs= stmt.executeQuery(VOpcao+" * from teste53 LEFT JOIN teste58 ON
(teste53.NUMERO_DA_VENDA=teste58.NUMERO_DA_VENDA)  ");
          while (rs.next()){
//VENDAS
            out.write(""+
"NUMERO_DA_VENDA:"+rs.getDouble("NUMERO_DA_VENDA")+"\n" + "");
            out.write(""+ "DATA:"+rs.getDate("DATA")+"\n" + "");
            out.write(""+
"CODIGO_DA_CONCESSIONARIA:"+rs.getDouble("CODIGO_DA_CONCESSIONA
RIA")+"\n" + "");
            out.write(""+
"CODIGO_DA_MONTADORA:"+rs.getDouble("CODIGO_DA_MONTADORA")+"\n
" + "");
            out.write(""+ "
CPF_DO_CLIENTE:"+rs.getDouble("CPF_DO_CLIENTE")+"\n" + "");
            out.write(""+ "CODIGOBARRAS:"+rs.getString("CODIGOBARRAS")+"\n" +
"");
            out.write(""+ "QUANTIDADE:"+rs.getDouble("QUANTIDADE")+"\n" + "");
           //FIM VENDAS
          }
```

```
//cliente@tccbd22
          jTextArea1.append("O numero de Registro para seleção é: "+QtReg+"\n");
           //recebe um statement de conexao
           inicio=System.currentTimeMillis();
  rs= stmt.executeQuery(VOpcao+" * from teste52 LEFT JOIN teste57 ON
(teste52.NUMERO_DA_VENDA=teste57.NUMERO_DA_VENDA)  ");
          while (rs.next()){
  //VENDAS
             out.write(""+
"NUMERO_DA_VENDA:"+rs.getDouble("NUMERO_DA_VENDA")+"\n" + "");
             out.write(""+ "DATA:"+rs.getDate("DATA")+"\n" + "");
             out.write(""+
"CODIGO_DA_CONCESSIONARIA:"+rs.getDouble("CODIGO_DA_CONCESSIONA
RIA")+"\n" + "");
             out.write(""+
"CODIGO_DA_MONTADORA:"+rs.getDouble("CODIGO_DA_MONTADORA")+"\n
" + "");
             out.write(""+ "
CPF_DO_CLIENTE:"+rs.getDouble("CPF_DO_CLIENTE")+"\n" + "");
             out.write(""+ "CODIGOBARRAS:"+rs.getString("CODIGOBARRAS")+"\n" +
"");
             out.write(""+ "QUANTIDADE:"+rs.getDouble("QUANTIDADE")+"\n" + "");
            //FIM VENDAS
           }
           fim=System.currentTimeMillis();
           rs.close();
           jTextArea1.append("Tempo inicio:"+inicio+"\n");
           jTextArea1.append("Tempo final:"+fim+"\n");
           t= fim - inicio;
           decimal = new DecimalFormat( "0.00" );
           time2= t/1000; //para achar em segundos o tempo
           decimal.format(time2);
           jTextArea1.append("\nO tempo gasto para Recuperação do(s) dado(s) na maquina
Lab0622 foi: "+time2+" segundos\n");
            //cliente@tccbd21
           jTextArea1.append("O numero de Registro para seleção é: "+QtReg+"\n");
           //recebe um statement de conexao
           inicio=System.currentTimeMillis();
  rs= stmt.executeQuery(VOpcao+" * from teste51 LEFT JOINteste56 ON
(teste51.NUMERO_DA_VENDA=teste56.NUMERO_DA_VENDA)  ");
           while (rs.next()){
```

```
//VENDAS
            out.write(""+
"NUMERO_DA_VENDA:"+rs.getDouble("NUMERO_DA_VENDA")+"\n" + "");
            out.write(""+ "DATA:"+rs.getDate("DATA")+"\n" + "");
            out.write(""+
"CODIGO_DA_CONCESSIONARIA:"+rs.getDouble("CODIGO_DA_CONCESSIONA
RIA")+"\n" + "");
            out.write(""+
"CODIGO_DA_MONTADORA:"+rs.getDouble("CODIGO_DA_MONTADORA")+"\n
" + "");
            out.write(""+ "
CPF_DO_CLIENTE:"+rs.getDouble("CPF_DO_CLIENTE")+"\n" + "");
            out.write(""+ "CODIGOBARRAS:"+rs.getString("CODIGOBARRAS")+"\n" +
"");
            out.write(""+ "QUANTIDADE:"+rs.getDouble("QUANTIDADE")+"\n" + "");
           //FIM VENDAS
          }
          fim=System.currentTimeMillis();
          rs.close();
          jTextArea1.append("Tempo inicio:"+inicio+"\n");
          jTextArea1.append("Tempo final:"+fim+"\n");
          t= fim - inicio;
          decimal = new DecimalFormat( "0.00" );
          time2= t/1000; //para achar em segundos o tempo
          decimal.format(time2);
          jTextArea1.append("\nO tempo gasto para Recuperação do(s) dado(s) na maquina
Lab0621 foi: "+time2+" segundos\n");

        stmt.close();
          jTextArea1.append("\nRegistros selecionados com sucesso na tabela");

          //salva no TXT
          out.write(""+ jTextArea1.getText() + "");
          out.close();
          //fim do TXT
      }
          catch (Exception e)
          {
          JOptionPane.showMessageDialog(null,"Impossível Recuperar os Dados, pelo
Seguinte Motivo:\n"+e,"Atenção",JOptionPane.WARNING_MESSAGE);
          }finally{
          }
      } //fim da opção select
```

```
 if (VOpcao== "delete")
    {
      try
        {
       jTextArea1.append("O numero de Registro para deletar é: "+QtReg+"\n");
       //recebe um statement de conexao
       Statement stmt= connection.createStatement();
       long timestamp=System.currentTimeMillis();
       double time,time2, soma;
       long inicio=System.currentTimeMillis();
       ResultSet rs= stmt.executeQuery(VOpcao+" from teste60");
       rs= stmt.executeQuery(VOpcao+" from teste55");
       long fim=System.currentTimeMillis();
       jTextArea1.append("Tempo inicio:"+inicio+"\n");
       jTextArea1.append("Tempo final:"+fim+"\n");
       long t= fim - inicio;
       DecimalFormat decimal = new DecimalFormat( "0.00" );
       time2= t/1000; //para achar em segundos o tempo
       decimal.format(time2);
       jTextArea1.append("\nO tempo gasto para Apagar o(s) dado(s) na maquina
Lab0625 foi: "+time2+" segundos\n");
       jTextArea1.append("\nRegistros selecionados com sucesso na tabela");
       //cliente@tccbd24
       jTextArea1.append("O numero de Registro para deletar é: "+QtReg+"\n");
       //recebe um statement de conexao
       inicio=System.currentTimeMillis();
       rs= stmt.executeQuery(VOpcao+" from teste59");
       rs= stmt.executeQuery(VOpcao+" from teste54");
       fim=System.currentTimeMillis();
       jTextArea1.append("Tempo inicio:"+inicio+"\n");
       jTextArea1.append("Tempo final:"+fim+"\n");
       t= fim - inicio;
       decimal = new DecimalFormat( "0.00" );
       time2= t/1000; //para achar em segundos o tempo
       decimal.format(time2);
       jTextArea1.append("\nO tempo gasto para Apagar o(s) dado(s) na maquina
Lab0624 foi: "+time2+" segundos\n");
       jTextArea1.append("\nRegistros selecionados com sucesso na tabela");
       //cliente@tccbd23
       jTextArea1.append("O numero de Registro para deletar é: "+QtReg+"\n");
       //recebe um statement de conexao
       inicio=System.currentTimeMillis();
```

```
            rs= stmt.executeQuery(VOpcao+" from teste58");
            rs= stmt.executeQuery(VOpcao+" from teste53");
            fim=System.currentTimeMillis();
            jTextArea1.append("Tempo inicio:"+inicio+"\n");
            jTextArea1.append("Tempo final:"+fim+"\n");
            t= fim - inicio;
            decimal = new DecimalFormat( "0.00" );
            time2= t/1000; //para achar em segundos o tempo
            decimal.format(time2);
            jTextArea1.append("\nO tempo gasto para Apagar o(s) dado(s) na maquina
Lab0623 foi: "+time2+" segundos\n");
            jTextArea1.append("\nRegistros selecionados com sucesso na tabela");

             //cliente@tccbd22
            jTextArea1.append("O numero de Registro para deletar é: "+QtReg+"\n");
            //recebe um statement de conexao
            inicio=System.currentTimeMillis();
             rs= stmt.executeQuery(VOpcao+" from teste57");
            rs= stmt.executeQuery(VOpcao+" from teste52");
            fim=System.currentTimeMillis();
            jTextArea1.append("Tempo inicio:"+inicio+"\n");
            jTextArea1.append("Tempo final:"+fim+"\n");
            t= fim - inicio;
            decimal = new DecimalFormat( "0.00" );
            time2= t/1000; //para achar em segundos o tempo
            decimal.format(time2);
            jTextArea1.append("\nO tempo gasto para Apagar o(s) dado(s) na maquina
Lab0622 foi: "+time2+" segundos\n");
            jTextArea1.append("\nRegistros selecionados com sucesso na tabela");

            //cliente@tccbd21
            jTextArea1.append("O numero de Registro para deletar é: "+QtReg+"\n");
            //recebe um statement de conexao
            inicio=System.currentTimeMillis();
rs= stmt.executeQuery(VOpcao+" from teste56");
            rs= stmt.executeQuery(VOpcao+" from teste51");
            fim=System.currentTimeMillis();
            jTextArea1.append("Tempo inicio:"+inicio+"\n");
            jTextArea1.append("Tempo final:"+fim+"\n");
            t= fim - inicio;
decimal = new DecimalFormat( "0.00" );
            time2= t/1000;
```

```
decimal.format(time2);
          jTextArea1.append("\nO tempo gasto para Apagar o(s) dado(s) na maquina
Lab0621 foi: "+time2+" segundos\n");
          jTextArea1.append("\nRegistros selecionados com sucesso na tabela");

          //
          connection.commit();
          stmt.close();
          jTextArea1.append("\nRegistros apagados com sucesso na tabela");

          //salva no TXT
          File outputFile = new File("Resposta.txt");
          FileWriter out = new FileWriter(outputFile);
          out.write(""+ jTextArea1.getText() + "");
          out.close();
          //fim do TXT
         }
        catch (Exception e)
          {
          //retorna o erro e pára
          JOptionPane.showMessageDialog(null,"Impossível Deletar os Dados, pelo
Seguinte Motivo:\n"+e,"Atenção",JOptionPane.WARNING_MESSAGE);
          }
          finally
          {

          }
      }
    }
```