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

CRISTIANO SEITI TAKANO

CONVERSOR DE BANCO DE DADOS ORACLE PARA POSTGRESQL

MARÍLIA, SP.

2005

CRISTIANO SEITI TAKANO

CONVERSOR DE BANCO DE DADOS ORACLE PARA POSTGRESQL

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

Orientador:
Prof. Dr. Edmundo Sérgio Spoto

MARÍLIA, SP.
2005

Cristiano Seiti Takano

“CONVERSOR DE BANCO DE DADOS ORACLE PARA POSTGRESQL”

Banca examinadora da monografia ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para a obtenção do Título de bacharel em Ciência da Computação.

Nota: _____(_____)

Orientador: Edmundo Sérgio Spoto _____

1º. Examinador: Ildeberto Aparecido Rodello _____

2º. Examinador: Maria Istela Cagnin _____

Marília, 02 de dezembro de 2005.

DEDICATÓRIA

Este trabalho é dedicado à minha família, por me apoiar durante o curso de graduação, sendo de grande importância, em todos os problemas e dificuldades que passamos ao longo desses anos. Agradeço especialmente minha mãe e irmã, que foram de grande importância nestes quatro anos de curso.

Muito obrigado.

AGRADECIMENTOS

Agradeço a todos os professores com quem tive contato durante a graduação, por me fornecerem todo o conhecimento necessário em suas respectivas áreas e em especial ao professor e doutor Edmundo Sérgio Spoto, por me orientar no desenvolvimento deste trabalho.

Agradeço a todos os meus familiares que me apoiaram, e que de alguma forma me ajudaram, para a realização deste trabalho.

Agradeço também a todos os amigos, que me ajudaram diretamente ou indiretamente para a conclusão deste trabalho.

Takano, Cristiano Seiti. Conversor de Banco de dados Oracle para PostgreSQL.
Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino “Eurípides Soares da Rocha”, Centro Universitário Eurípides de Marília – UNIVEM. Marília, 2005.

RESUMO

Neste projeto se propõe a implementação de um conversor de dados do Oracle para o PostgreSQL e um tutorial de instruções SQL dos bancos Oracle e PostgreSQL. O desenvolvimento foi baseado nas linguagens HTML, DHTML, PHP (*Hypertext Preprocessor*), javascript e uma base de dados MYSQL, para a manipulação dos registros do tutorial.

No desenvolvimento do conversor, utilizou-se o SGBD Oracle para a extração das tabelas e seus atributos, chaves primárias e estrangeiras e registros contidos nas tabelas e posteriormente fazer a conversão para o PostgreSQL.

O planejamento do tutorial foi estruturado de forma que se pode inserir, alterar e modificar os dados de qualquer lugar, utilizando um navegador de Internet (para fazer isto se deve ter *login* e senha do sistema).

Palavras chave: Banco de Dados, Oracle, PostgreSQL, PHP.

Takano, Cristiano Seiti. Conversor de Banco de dados Oracle para PostgreSQL.
Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino “Eurípides Soares da Rocha”, Centro Universitário Eurípides de Marília – UNIVEM. Marília, 2005.

ABSTRACT

In this project there is proposed the implementation of a converter of data of the Oracle for the PostgreSQL and a tutorial of instructions SQL of the database above cited. The development will be based on languages HTML, DHTML, PHP (Hypertext Preprocessor), Javascript and a database MYSQL for the tutorial one.

In the development of the converter, Oracle was used the SGBD for the extraction of the charts and his attributes, primary and foreign keys and registers contained in the charts and subsequently to do the conversion for the PostgreSQL.

The projection of the tutorial was structured in such a way that it is possible to insert, to alter and to modify the data of any place, using a navigator of Internet (to do this it is necessary to have login and sign of the system).

Keywords: Database, Oracle, PostgreSQL, PHP.

LISTA DE TABELAS

Tabela 3.1 - Mostra a comparação dos Bancos de Dados 21
Tabela 4.1 - Apresenta os tipos de variáveis do PHP 26
Tabela 4.2 - Apresenta a comparação dos tipos de dados do Oracle e PostgreSQL 27

LISTA DE QUADROS

Quadro 3.1 - Mostra a criação de tabelas 23
Quadro 3.2 - Mostra a criação de tabelas 23
Quadro 3.3 - Mostra a inserção das chaves 23
Quadro 3.4 - Mostra o comando de inserção de registros 24
Quadro 3.5 - Mostra os comandos de alteração dos campos 24
Quadro 4.1 - Comandos para extrair os atributos 28
Quadro 4.2 - Mostra o comando SQL para a criação da tabela tutorial 30
Quadro 4.3 - Mostra os comandos do mysql utilizados. 30
Quadro 5.1 - Tabela DEPT 32
Quadro 5.2 - Tabela EMP 32
Quadro 5.3 - Tabela BONUS 32
Quadro 5.4 - Tabela SALGRADE 33
Quadro 5.5 - Parte do código de conversão de tipos 35
Quadro 5.6 - Mostra a instrução SQL do Oracle 36
Quadro 5.7 - Mostra o resultado da conversão SQL 36

LISTA DE FIGURAS

Figura 2.1 - Representação de um Sistema de Banco de Dados..... 16
Figura 4.1 - Mostra o resultado do comando SQL 28
Figura 4.2 - Mostra o resultado da consulta de chaves..... 29
Figura 4.3 - Apresenta o comando e o resultado do registro do Oracle 29
Figura 5.1 - Mostra o modelo de Dados Lógico..... 31
Figura 5.2 - Mostra o modelo de Dados Físico. 31
Figura 5.3 - Mostra os dados extraídos do Oracle..... 33
Figura 5.4 - Mostra o Resultado da conversão do Oracle para o PostgreSQL. 34
Figura 5.5 - Apresenta o conversor de comandos SQL..... 35
Figura 5.6 - Mostra a página principal 37
Figura 5.7 - Mostra os campos do tutorial..... 37
Figura 5.8 - Mostra o editor de registros 38
Figura 5.9 - Mostra o resultado da pesquisa..... 38

LISTA DE ABREVIATURAS E SIGLAS

ARO - *Army Research Office*

BD - Banco de Dados

CGI - *Common Gateway Interface*

DARPA - *Defense Advanced Research Projects Agency*

DDL - *Data Definition Language* (Linguagem de Definição de Dados)

DML - *Data Manipulation Language* (Linguagem de Manipulação de Dados)

MB - *Megabyte*

MER - Modelo de Entidades Relacionamento

MHz - *Megahertz*

NSF - *National Science Foundation*

ODBC - *Open Database connectivity*

ODBMS - *Object Database Management Systems*

OO - *Object-Oriented* (Orientação a Objetos)

ORDBMS - *Object-Relational Database Managenent System*

PC - *Personal Computer*

PHP - *Hypertext Preprocessor*

QUEL - *Query Language*

RAM - *Random Acess Memory*

RDM - *Relational Data Modeling* (Modelo de Dados Relacional)

SGBD - Sistema de Gerenciamento de banco de Dados

SGBDR - Sistema de Gerenciamento de banco de Dados Relacional

SQL - *Structured Query Language*

SEQUEL - Structured English Query Language

TEMP - Arquivo temporário

QBE - *Query-By-Example*

UCB - *University of Califórnia, Berkeley*

WWW - *World Wide Web*

SUMÁRIO

1. CONSIDERAÇÕES INICIAIS 13

1.1 Introdução 13

1.2 Objetivos 14

1.2.1 Objetivo Geral 14

1.2.2 Objetivos Específicos 15

1.3 Organização deste Trabalho 15

2. DEFINIÇÃO DE BANCO DE DADOS - CONCEITOS E TERMINOLOGIA 16

2.1 Definição de Banco de Dados 16

2.2 Banco de Dados Relacional 17

2.3 PostgreSQL 17

2.4 Sistema Gerenciador de Banco de Dados Oracle 18

3. COMANDOS DE SQL 20

3.1 Histórico de SQL 20

3.2 Comparação dos Bancos de Dados 21

3.3 Comparação de Alguns Comandos SQL do Oracle e PostgreSQL 22

4. MATERIAIS E MÉTODOS 25

4.1 Linguagem PHP 25

4.2 Comparação dos Tipos de Dados Suportados pelos SGDBs 26

4.3 Extraindo Tabelas e Tipos de Dados do Oracle 27

4.3 Desenvolvimento do Tutorial 29

5. ESTUDO DE CASO 31

5.1 Resultado Obtido do Conversor 33

5.2 Resultados Obtidos do Tutorial e da Pesquisa 36

6. CONCLUSÃO E TRABALHOS FUTUROS 39

6.1 Conversor de Banco de dados Oracle para PostgreSQL 39

6.2 Trabalhos Futuros 39

REFERÊNCIAS BIBLIOGRÁFICAS 41

ANEXO A - INSTALAÇÃO DO POSTGRESQL 43

ANEXO B - INSTALAÇÃO DO ORACLE 49

ANEXO C - PRINCIPAIS COMANDOS SQL DO POSTGRESQL 53

ANEXO D - PRINCIPAIS COMANDOS SQL DO ORACLE 65

1. CONSIDERAÇÕES INICIAIS

1.1 Introdução

Os Sistemas de Gerenciamento de Banco de Dados (SGBDs) são ferramentas para o armazenamento, recuperação e manutenção de dados, possibilitando uma maneira mais eficaz de gerenciamento. Os SGDBs utilizados na conversão são o Oracle e PostgreSQL, sendo que para o primeiro é necessário a obtenção de licença para uso comercial e, para o segundo não, pois o banco é *open-source* (código aberto), possibilitando que programadores alterem seu código de forma a corrigir eventuais erros (*bugs)*.

De acordo com SILBERSCHATZ (1999), as linguagens comerciais (utilizadas) para fazer uma consulta nos SGBDs são SQL (*Struct Query Language*) e QBE (*Query-By-Example*). A mais utilizada é a SQL, que utiliza uma combinação de construtores em álgebra e cálculo relacional. Além de ser uma linguagem de consulta, ela pode ser usada para fazer uma definição de estruturas de dados, fazer modificações no banco de dados e especificar as restrições de segurança. O QBE tem como base o cálculo relacional de domínios (conjunto de valores permitidos).

As linguagens declarativas SQL e QBE deverão ser escritas conforme os padrões e, cada usuário poderá escolher a melhor forma de fazer uma consulta no SGBD, consulta essa que deverá estar otimizada para conseguir melhor desempenho no resultado. Para isso o domínio dessa linguagem é fundamental.

Existe um produto comercial da empresa "*dbExperts*", que disponibiliza um pacote de software onde pode ser encontra uma conversão de banco do Oracle para o PostgresSQL na plataforma Linux. A empresa não disponibiliza o que é feito e como é feito está conversão, assim impossibilitando a comparação entre o projeto proposto e o pacote citado.

O projeto a ser apresentado nesta monografia terá a finalidade de ajudar programadores que precisam fazer uma conversão do Oracle para o PostgreSQL e, um tutorial explicativo de fácil navegação e obtenção da informação, exibindo exemplos dos comandos SQL.

1.2 Objetivos

Os bancos de dados *open-source* (código aberto) têm um baixo custo de implantação em pequenas, médias e grandes empresas, mas se a empresa tiver muitas *triggers*[1] sendo utilizadas no banco de dados o custo pode aumentar consideravelmente na migração para outro SGBD, e também por causa da incompatibilidade nas peculiaridades da linguagem de cada SGBD.

O objetivo deste trabalho é fazer uma extração das tabelas e atributos do Oracle e convertê-las para o PostgreSQL (*open-source*), serão citados alguns bancos *open-source*: PostgreSQL, Mysql, Firebird, HSQL *Database Engine*, Berkeley DB, entre outros. Neste trabalho, será utilizado o SGBD PostgreSQL e Oracle, que serão estudados para a confecção de um tutorial, dos principais comandos de SQL dos Bancos, a implantação dos bancos e algumas dicas de configuração dos SGBDs.

1.2.1 Objetivo Geral

Apresentar um estudo de Banco de Dados de código fonte livre PostgreSQL, e um BD comercial Oracle, nestes SGBDs são utilizados a arquitetura Cliente/Servidor e definição de modelo de dados Objeto-Relacional. Estes sistemas de banco de dados serão utilizados para a análise, comparar as principais funcionalidades de PL/SQL dos SGBDs, e após esta verificação, foi implementado a conversão de tabelas do Oracle para o Banco de Dados PostgreSQL, avaliando as funcionalidades dos comandos e dos tipos de dados, que serão utilizados para o tutorial.

Com este tutorial, o usuário terá condições de implantar os SGBD, e ter um material de consulta dos principais comandos PL/SQL.

---

[1] Conjunto de comandos criados no próprio banco de dados, visando simplificar os programas que o utilizam. Pode-se definir que os comandos são executados quando o banco de dados realiza alguma operação, por exemplo, inserção, alteração ou exclusão de dados.

1.2.2 Objetivos Específicos

- Implantação dos Bancos de Dados (PostgreSQL e Oracle).

- Descrever os principais comandos SQL dos Bancos de Dados PostgreSQL e Oracle, instrumentos de pesquisa deste trabalho;

- Fazer a conversão das principais funções de SQL do Oracle para o PostgreSQL.

- Escrever um tutorial objetivo e de fácil entendimento para pessoas interessadas no assunto.

1.3 Organização deste Trabalho

Para o melhor entendimento deste trabalho, será apresentada uma amostra dos assuntos relacionados a cada capítulo, que serão mostrados da seguinte maneira.

Capítulo 2 apresenta uma definição do SGBDs e uma definição de BD relacional, uma explicação dos Bancos Oracle e PostgreSQL.

Capítulo 3 estudo dos principais comandos SQL dos BDs e seus atributos.

Capítulo 4 apresenta a linguagem e os métodos utilizados para a implementação deste trabalho.

Capítulo 5 apresenta um estudo de casos. No Capítulo 6 são apresentados os resultados obtidos deste trabalho. E finalizando no Capítulo 7 são apresentados as conclusões e os trabalhos futuros.

2. DEFINIÇÃO DE BANCO DE DADOS - CONCEITOS E TERMINOLOGIA

2.1 Definição de Banco de Dados

Segundo Chu (1943): “Banco de Dados (BD) é um conjunto de dados estruturados de maneira adequada de forma que pode ser utilizado com eficiência por uma diversidade de aplicações dentro de uma organização”.

O Sistema de Banco de Dados (SGBD) em uma organização corresponde a um conjunto de arquivos de dados de toda organização, armazenadas em algum tipo de dispositivo magnético, sendo manipulado por um conjunto de programas, representada na Figura 2.1.

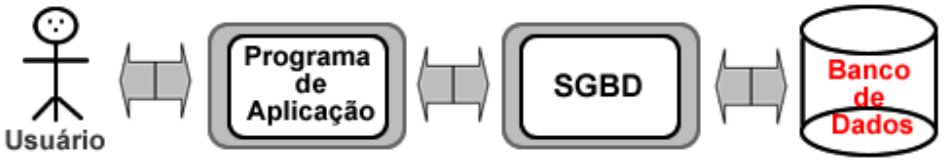


Figura 2.1 - Representação de um Sistema de Banco de Dados.

Estes programas manipulam o Banco de dados, fazendo alterações (ALTER TABLE), inserções (INSERT), exclusões (DELETE) dos dados, assim como processos de cálculo, regravação de informações mais elaboradas, operações de pesquisa mais complexas para os níveis de controle gerencial. O sistema de Banco de Dados fornece informações para análise e simulação de alternativas no processo de planejamento empresarial.

A construção de um Bando de Dados só seria possível se todas as informações estivessem em um Banco integrado. Para a realização das consultas de diversas áreas da empresa, teria que ter um meio físico que fizesse a comunicação com terminais de entrada e saída, espalhadas em vários setores da empresa com níveis hierárquicos diferentes, assim teria a possibilidade de acesso remoto ao Banco de Dados integrado.

2.2 Banco de Dados Relacional

O RDM - Modelo de Dados Relacional trata-se de um modelo criado na década de 70 pelo pesquisador da IBM Dr. E. F. Codd, cujo propósito era representar os dados de forma mais simples, através de um modelo matemático de conjuntos de tabelas inter-relacionadas. Este modelo torna os bancos de dados mais flexíveis, tanto na forma de representar as relações entre os dados, como na tarefa de modificação de sua estrutura, sem ter que reconstruir todo o banco de dados. (SILBERSCHATZ, 1999)

Os novos modelos de banco de dados com tecnologia chamada de orientado a objetos são considerados uma forte tendência para o estabelecimento de um modelo para novas tecnologias de banco de dados.

Como alternativa, os fabricantes de SGBD adotaram a medida de incorporar características OO aos modelos relacionais, criando assim os modelos de banco de dados objeto-relacional (ORDBMS - *Object-Relational Database Managenent System*).

Atualmente o modelo de dados Objeto-Relacional está sendo implementado na maioria dos produtos de SGBD. Acredita-se que o uso destes produtos será cada vez mais difundido em relação aos ORDBMS devido ao sucesso dos já consolidados (Oracle e PostgreSQL), segundo a documentação PostgreSQL 8.0.0, 2005 e ORACLE.

2.3 PostgreSQL

De acordo com a documentação do PostgreSQL 8.0.0 (2005), ele é um Sistema de Gerenciamento de Banco de Dado objeto-relacional (SGBDOR), baseado no POSTGRES, Versão 4.2, desenvolvido no Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley. O Projeto POSTGRES, liderado pelo professor Michael Stonebraker, foi patrocinado pelas instituições: *Defense Advanced Research Projects Agency* (DARPA), *Army Research Office* (ARO), *National Science Foundation* (NSF) e ESl, Inc.

O POSTGRES foi pioneiro em muitos conceitos de objeto-relacional, e alguns bancos de dados, disponibilizam estes recursos, suportando um modelo de dados que consiste em uma coleção de relações com nome, contendo atributos de um tipo específico. Os tipos possíveis incluem número de ponto flutuante, inteiros, cadeia de caracteres, monetário e data, segundo a documentação do PostgreSQL 8.0.0 (2005).

O modelo relacional se estabeleceu com o primeiro modelo de dados para aplicações comerciais, substituindo os modelos anteriores com sucesso em partes devido a sua "simplicidade", entretanto esta simplicidade tornou-se a implementação de certas aplicações muito difícil (SILBERSCHATZ, 1999).

O PostgreSQL oferece um poder adicional, devido à incorporação dos conceitos de herança, tipos de dados e funções. Os usuários podem facilmente estender o sistema sendo que outras funcionalidades fornecem poder e flexibilidade adicionais como *constraints*, *triggers*, *rules*, integridade da transação. Estas colocam o PostgreSQL dentro da categoria de banco de dados objeto-relacional. (Documentação do PostgreSQL 8.0.0 , 2005)

Com mais de uma década de desenvolvimento pos trás, o PostgreSQL é o mais avançado banco de dados de código aberto disponível em qualquer lugar, oferecendo controle de concorrência multi-versão, suportando praticamente todas as construções do SQL (incluindo sub-consultas, transações, tipos definidos pelo usuário e funções) e dispondo de um amplo conjunto de ligações com linguagens procedurais (incluindo C, C++, Java, Pearl, Tcl e Python).

2.4 Sistema Gerenciador de Banco de Dados Oracle

De acordo com Morelli (2002), "desde 1979, quando foi lançado o primeiro SGDBR comercial, a Oracle notabilizou-se por nortear os rumos do mercado de BDs Corporativos". Em julho de 2001 foi lançado o Oracle 9i de seu SGDBR, este foi o primeiro a permitir acesso a dados de barramento de 64 bits.

O Oracle é um SGBD cliente/servidor de alta performance, que pode ser instalado em diferentes plataformas como UNIX, LINUX ou WINDOWS. Este banco pode suportar um

grande volume de dados em um ambiente multi-usuário, com acessos simultâneos e sem perda no desempenho.

A linguagem SQL pode ser inserida em ambientes como o SQL*Plus, ou o SQL*Plus *Worksheet*, sendo que neles pode-se fazer a manipulação de comandos PL/SQL. O Oracle possibilita a manipulação de ***triggers, functions, procedures*** e possui um mecanismo de integridade referencial, que assegura que os dados não tenham duplicidade em uma mesma tabela.

Este banco disponibiliza muitas ferramentas para a administração das tabelas e usuários de forma gráfica facilitando o manuseio. A ferramenta conhecida como *Oracle Enterprise Manager Console* é utilizada para interagir com o Banco de Dados e Gerenciar os esquemas (tabelas, índices, *views*, etc.) e usuários. Para se conectar no Oracle utilizando cliente/servidor, deve-se configurar o *Listener* e a configuração de rede local da máquina, que se encontra em *Configuration* and *Migration Tools/Net Configuration Assistant*.

Uma comparação dos bancos será mostrada no Capítulo 3.

3. COMANDOS DE SQL

Neste capítulo, serão mostrados um breve histórico de SQL, a comparação dos bancos e a comparação de alguns comandos SQL dos Bancos estudados neste trabalho.

3.1 Histórico de SQL

Segundo Chu (1943), os comandos SQL têm se tornado o padrão dos bancos de dados relacionais. A versão original foi desenvolvida pela IBM no laboratório de pesquisa de San José (Hoje Centro de Pesquisa Almadem). Essa linguagem se chamava Sequel, implementada nos anos 70 como parte do projeto do Sistema R. Desde então a linguagem foi evoluindo e passou a se chamar SQL (*Structured Query Language* - Linguagem de Consulta Estruturada).

No ano de 1986, publicou-se o padrão para SQL, chamada de SQL-86. Os responsáveis foram *Americam National Standards Institute* (ANSI) e a *International Standards Organization* (ISO). Outras empresas publicaram seus próprios padrões, como a IBM e a *Sytems Application Architecture* em 1987. Foi publicada em 1989 uma extensão para o padrão SQL, SQL-89. Os sistemas de BD típicos dão suporte pelo menos aos recursos dispostos na SQL-89. A versão em uso do padrão ANSI/ISO SQL é o padrão SQL-92 e o SQL-99. (SILBERSCHATZ, 1999).

A Linguagem SQL incorpora diversas linguagens:

- **Linguagem de Definição de Dados** (*Data-definition Language* - DDL). Essa linguagem proporciona comandos para a definição de esquemas de relações, exclusão de relações, criação de índices e modificação nos esquemas de relação.
- **Linguagem Interativa de Manipulação de Dados** (*Data-manipulation Language*-DML). Essa linguagem abrange comandos de consulta baseada tanto na álgebra relacional quanto no cálculo de tuplas (linhas ou elementos da relação), englobando ainda os comandos de inserção, de exclusão e de alteração de tuplas no BD.

- **Incorporação DML** (*Embedded* DML). A forma a qual o comando SQL incorporado foi projetado para a aplicação em linguagens de programação de uso geral, como PL/I, Cobol, Pascal, Fortram e C.
- **Definição de Visões**: com uso da DDL, a SQL possui comandos para a definição de visões (tabela virtual na qual os dados não estão fisicamente armazenados).
- **Autorização**: possui os comandos para especificação de direitos de acesso à relação de visões.
- **Integridade**: possui comandos para especificação de regras de integridade para que os dados ao serem armazenados ou atualizados não violem as regras de integridade.
- **Controle de Transações**: comandos para definição de iniciação e finalização de transações. Algumas das implementações permitem explicitar bloqueios de dados para o controle de concorrência.

3.2 Comparação dos Bancos de Dados

Na Tabela 3.1 são mostradas as comparações dos bancos de dados e dos sistemas operacionais suportados (Comparativo, 2003).

Tabela 3.1 - Mostra a comparação dos Bancos de Dados

	PostgreSQL	**Oracle**	**MS SQL**	**MySQL**	**Interbase**
Open Source	✔			✔	
Plataformas					
Linux	✔	✔		✔	✔
FreeBSD	✔			✔	
Windows	✔	✔	✔	✔	✔
Sun Solaris	✔	✔			
Mac OS X	✔				

Tabela 3.1 - Mostra a comparação dos Bancos de Dados (Continuação)

	PostgreSQL	Oracle	MS SQL	MySQL	Interbase
HP UX	✔	✔			
Características					
ACID	✔	✔			
Stored Procedures / Triggers	✔	✔	✔		✔
Transações Concorrentes	✔	✔			
ANSI SQL 99	✔				
ANSI SQL 92	✔	✔	✔		✔
Sem limites de usuários	✔			✔	
Integridade Referencial	✔	✔	✔		✔
Transações	✔	✔	✔		✔
ODBC Free	✔	✔	✔		

3.3 Comparação de Alguns Comandos SQL do Oracle e PostgreSQL

No Oracle, de acordo com FERNANDES (2002) e MORELLI (2002), os comandos de SQL devem ser inseridos no SQLPlus ou SQL*Plus *Worksheet.* Essas ferramentas instaladas com o banco de dados executaram todas as classes de comandos SQL. Possuem comandos padrão do SQL e que são desenvolvidos para SGBD Oracle. O padrão suportado é o ANSI/ISO SQL-92 (ANSI – ***American National Standards Intitute***, ISO – ***International Organization of Standards***). (ORACLE)

No PostgreSQL, segundo a documentação do PostgreSQL 8.0.0 (2005), “os comandos em SQL são formados por uma seqüência de termos (*Tokens*), terminados por ponto-e-vírgula (“;”). Um termo pode ser uma palavra chave, um identificador entre aspas, um literal ou constante, ou um caractere especial”. Os comandos SQL deverão ser inseridos no pgAdmin, que é instalado junto com o PostgreSQL. O pgAdmin é um ambiente gráfico para administrar o SGBDs. Os padrões suportados são ANSI SQL-89, SQL-92 e SQL-99.

Um exemplo de como criar um banco de dados: descrito no Quadro 3.1

Oracle	PostgreSQL
CREATE DATABASE nome_database (DATABASE_ID database_id, DATABASE_SIZE max_bytes, EXTENT_SIZE npages);	CREATE DATABASE nome_db [[WITH] [OWNER [=] dono_do_banco_de_dados] [TEMPLATE [=] modelo] [ENCODING [=] codificação] [TABLESPACE [=] espaço_de_tabelas]] ;

Quadro 3.1 - Mostra a criação de tabelas

Para a criação de tabelas usam-se os comandos representados no Quadro 3.2.

Oracle	PostgreSQL
CREATE TABLE nome_tabela (item1 NUMBER(4), item1 VARCHAR2(30) NOT NULL,);	CREATE TABLE nome_tabela (item1 integer, item2 varchar(20),);

Quadro 3.2 - Mostra a criação de tabelas

Para inserir chaves primárias e estrangeiras após a criação das tabelas, utilizam-se os comandos ilustrados no Quadro 3.3.

Oracle	PostgreSQL
ALTER TABLE nome_tabela ADD **(**PRIMARY KEY (nome_campo) **)**; ALTER TABLE nome_tabela ADD **(**FOREIGN KEY (nome_campo) REFERENCES nome_tabela_referente **)**;	ALTER TABLE nome_tabela ADD PRIMARY KEY (nome_campo); ALTER TABLE nome_tabela ADD FOREIGN KEY (nome_campo) REFERENCES tabela_referenciada;

Quadro 3.3 - Mostra a inserção das chaves

Existem pequenas diferenças nestes comandos, no Oracle são utilizados parênteses após o “ADD” e no final da instrução, já no PostgreSQL, não são necessário.

Para inserir registros nas tabelas utiliza-se o comando que será mostrado no Quadro 3.4.

Oracle	PostgreSQL
INSERT INTO nome_tabela VALUES (‘valor_campo1’, ‘valor_campo2’, 123 ...); OU INSERT INTO nome_tabela (nome_campo1, nome_campo2, nome_campo3 ...) VALUES (‘valor_campo1’, ‘valor_campo2’, 123, ...);	INSERT INTO nome_tabela VALUES (‘valor_campo1’, ‘valor_campo2’, 123 ...); OU INSERT INTO nome_tabela (nome_campo1, nome_campo2, nome_campo3 ...) VALUES (‘valor_campo1’, ‘valor_campo2’,123, ...);

Quadro 3.4 - Mostra o comando de inserção de registros

Os comandos para inserir registros nos bancos são idênticos.

Para fazer uma modificação nas tabelas utiliza-se o ALTER TABLE. Com este comando podemos: **1.** adição dos campos; **2.** remover um campo; **3.** alterar a senha de um usuário, que serão descritos no Quadro 3.5.

Oracle	PostgreSQL
1. ALTER TABLE nome_tabela **ADD** (nome_campo1 CHAR(10) NULL); **2.** ALTER TABLE nome_tabela **DROP** (nome_campo); **3.** ALTER USER nome_usuario **IDENTIFIED BY** nova_senha;	**1.** ALTER TABLE nome_tabela **ADD COLUMN** nome_campo tipo_dado; **2.** ALTER TABLE nome_tabela **DROP COLUMN** nome_campo; **3.** ALTER USER nome_usuario **WITH PASSWORD** 'nova_senha';

Quadro 3.5 - Mostra os comandos de alteração dos campos

4. MATERIAIS E MÉTODOS

Neste Capítulo são apresentadas as ferramentas desenvolvidas para a conversão do banco de dados e o tutorial. Na confecção deste trabalho utilizou-se a linguagem PHP e os SGDBs apresentados no capítulo anterior.

Iniciaremos este capítulo apresentando uma breve descrição da linguagem utilizada e, em seguida, o estudo do dicionário de dados dos SGBDs e uma comparação para que possamos fazer a conversão correta do banco PostgreSQL 8.0 e Oracle 9i 9.2.0.1.0.

4.1 Linguagem PHP

Segundo Soares (2002), o PHP-*Hypertext Preprocessor* (pré-processador de hipertexto) é uma poderosa linguagem de programação de *script open source*, multiplataforma, que fornece suporte a vários bancos de dados, tendo suporte para linguagem estrutural ou orientada a objeto, implementado no PHP 5.

A linguagem é extremamente simples, porém muitos recursos e funções. Se pode desenvolver aplicativos de *desktop* do lado do cliente com PHP-GTK (informações em http://gtk.php.net/ ou http://www.php-gtk.org.br/).

O PHP pode ser utilizado na maioria dos sistemas operacionais e servidores existentes.

Os *scripts* podem ser misturados com o HTML, tornando-se fácil o desenvolvimento de páginas dinâmicas, que possuem delimitadores de início e fim, como por exemplo: **<?php** [*código]* **?>**. Estes comandos rodam no servidor retornando apenas o resultado para o usuário, que não saberá a linguagem utilizada (PHP, ASP, etc). O PHP não se limita a trabalhar somente com HTML, também possibilita funções para geração de imagens, arquivos PDF e animações *flash*, criando-as dinamicamente. É possível criar qualquer padrão texto como DHTML e outros arquivos XML, podendo gerar padrões de arquivos e salvá-los em um sistema, entre outros recursos disponibilizados no MANUAL DO PHP (2005).

Os tipos de variáveis do PHP ilustrada na Tabela 4.1:

Tabela 4.1 - Apresenta os tipos de variáveis do PHP

Nome	Tamanho	Faixa/Precisão	Descrição
Integer	32 bits	decimal (base 10), hexadecimal (base 16) ou octal (base 8)	utilizado para números inteiros
Float	64 bits	precisão de 14 decimais digitais	utilizado para números reais
String	cada string tem 1 byte	256 caracteres	utilizado para string de caracteres
Boolean	-	TRUE ou FALSE	utilizado para valores verdadeiros ou falsos
Array	-	-	utilizado para armazenar vários itens de dados do mesmo tipo
Object	-	-	utilizado pra armazenar instâncias de classes
Resourse	-	-	uma variável especial
Null	-	-	representa que a variável não tem valor

4.2 Comparação dos Tipos de Dados Suportados pelos SGDBs

A comparação entre os tipos de dados do Oracle e PostgreSQL será importante para que não ocorra a perda da integridade dos dados e, manter os tipos de dados compatíveis entre os SGBDs. Isto poderá ser visto no Apêndice B.

Na Tabela 4.2 é apresentado uma comparação dos dados Oracle/PostgreSQl..

Tabela 4.2 - Apresenta a comparação dos tipos de dados do Oracle e PostgreSQL

Oracle	PostgreSQL	Descrição
CHAR(tam)	CHAR(tam)	String de caracteres de tamanho (tam) fixo. O máximo é 255 e o default é 1.
VARCHAR2(tam)	VARCHAR(tam)	String de tamanho (tam) máximo 2000 bytes.
LONG	TEXT	Caracteres de dados maiores que 2 gigabytes, no PosgreSQL o comprimento de variável não limitado.
CLOB	TEXT	Caracteres de tamanho variável de no máximo é 4 gigabytes.
BLOB	BYTEA	Binário com no Maximo 2 gigabytes.
NUMBER(p,s)	NUMERIC(p,s)	Número de precisão (tamanho p), variando de 1 a 38 e escala de (decimais s) variando de -84 a 127.
FLOAT	NUMERIC	Numéricos não exatos de precisão variável. Numero de precisão decimal de até 15 dígitos 8 bytes respectivamente.
DOUBLE PRECISION	DOUBLE PRECISION	Precisão de 6 dígitos decimais.
RAW(tam)	BYTEA	Dados binários brutos (raw) de tamanho variável. Um tamanho (tam) deve ser especificado, e o máximo permitido é de 4 bytes.
LONG RAW(tam)	BYTEA	Dados binários brutos (raw) de tamanho (tam) variável até 2 gigabytes.
DATE	DATE	Datas entre 1 de janeiro de 4712 AC e 31 de dezembro de 4712 DC.
TIMESTAMP	TIMESTAMP	Data e hora

Após esta comparação entre os diferentes tipos de dados, foram confeccionados algoritmos que farão a conversão, assegurando que os tipos sejam compatíveis. Conforme demonstrado no resultado do conversor no Capítulo 5.

4.3 Extraindo Tabelas e Tipos de Dados do Oracle

Nesta etapa, foram estudados uma forma de recuperação das tabelas e seus respectivos atributos no Oracle. Para isso, foram confeccionados comandos SQL para acessar a base de dados e conseguir recuperar os atributos, ilustrados no Quadro 4.1.

```
// Comando para extrair as tabelas no Oracle
SELECT tname FROM tab ORDEY BY tname asc;
// Para encontrar os campos e atributos utilizamos as funções que os recupera
   for ($i = 1; $i <= $ncols; $i++) {
       $column_name  = OCIColumnName ($sql2, $i);
       $column_type   = OCIColumnType ($sql2, $i);
       $column_size    = OCIColumnSize ($sql2, $i);
 // Comando para extrair as chaves PK e FK
SELECT a.constraint_name as CONST_NAME, a.TABLE_NAME as
TAB_NAME, b.column_name AS COLUM_NAME
FROM user_constraints a, user_cons_columns b
WHERE a.constraint_name = b.constraint_name;
```

Quadro 4.1 - Comandos para extrair os atributos

O resultado da execução do comando que mostra as tabelas contidas na instalação do Oracle está ilustrado na Figura 4.1.

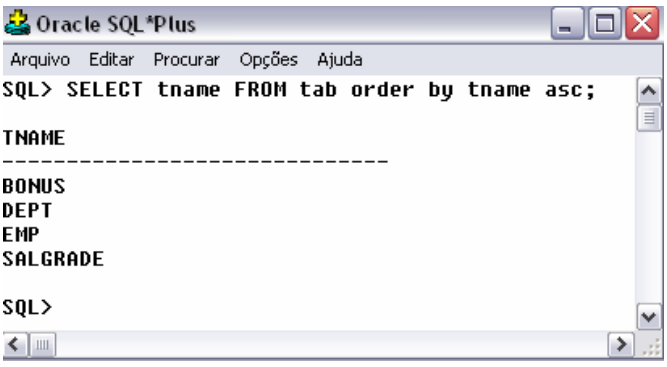


Figura 4.1 - Mostra o resultado do comando SQL

O resultado das chaves primária e estrangeira das tabelas é apresentado na Figura 4.2.

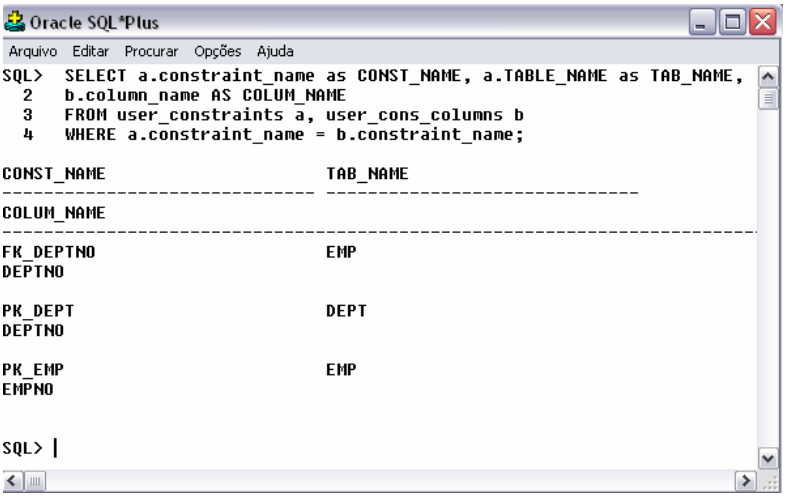


Figura 4.2 - Mostra o resultado da consulta de chaves.

Para a extração dos registros, foi confeccionado um comando em SQL para consultar as tabelas e posteriormente, formatar os registros para serem executadas no PostgreSQL. Na Figura 4.3 é mostrada a consulta de uma tabela e o seu resultado (no Oracle).

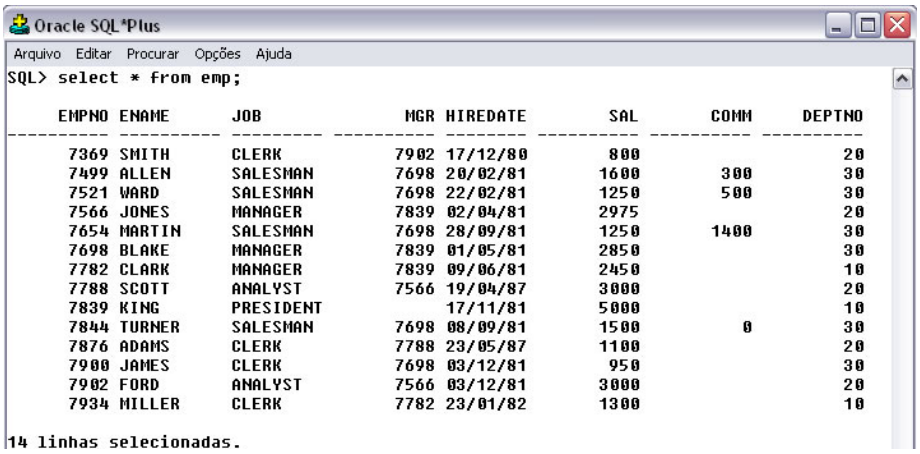


Figura 4.3 - Apresenta o comando e o resultado do registro do Oracle

4.3 Desenvolvimento do Tutorial

Criou-se uma tabela na base de dados Mysql para fazer o tutorial, inserindo e alterando os registros *online*. No Quadro 4.2 é mostrada a criação da tabela.

```
CREATE TABLE `tutorial` (
`codigo` INT( 5 ) NOT NULL AUTO_INCREMENT ,
`tutor_titulo` VARCHAR( 30 ) NOT NULL ,
`tutor_chave` VARCHAR( 30 ) NOT NULL ,
`tutor_postgresql` EXT NULL ,
`tutor_oracle` TEXT NULL,
PRIMARY KEY ( `codigo` ) )
) COMMENT = 'Tabela do Tutorial';
```

Quadro 4.2 - Mostra o comando SQL para a criação da tabela tutorial

Para efetuar a manipulação dos registros foram criados comandos em SQL, que estão ilustradas no Quadro 4.3.

```
1 SELECT * FROM tutorial where código;
2
3 INSERT INTO tutorial (tutor_titulo, tutor_chave,
4 tutor_postgresql, tutor_oracle) VALUES ('$tutor_titulo',
5 '$tutor_chave', '$tutor_postgresql', '$tutor_oracle')";
6
7 UPDATE tutorial SET tutor_titulo='$titulo', tutor_chave =
8 '$chave', tutor_postgresql = '$pos', tutor_oracle='$ora'
9 where codigo = '$codigo';
```

Quadro 4.3 - Mostra os comandos do mysql utilizados.

No Capítulo 5 será mostrado o estudo de caso.

5. ESTUDO DE CASO

O estudo de caso do banco de dados Oracle utilizará a ferramenta ERWIN para fazer o modelo de dados lógico e físico. A implementação do modelo de dados foi feita a partir da instalação do Oracle, que ao ser instalado, criou uma base de dados, ilustrada na Figura 5.1 e Figura 5.2.

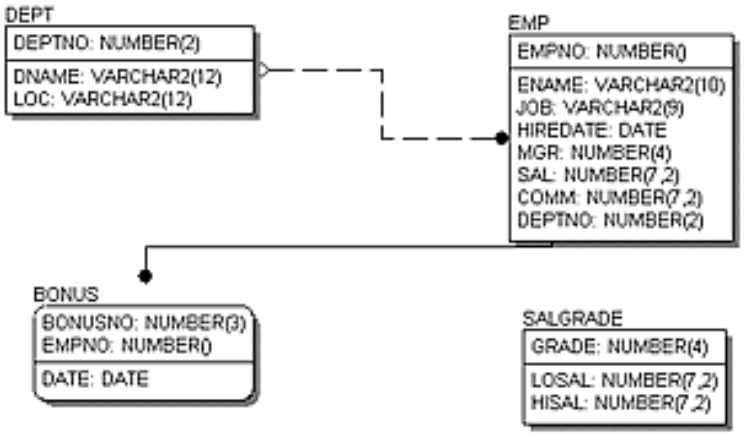


Figura 5.1 - Mostra o modelo de Dados Lógico.

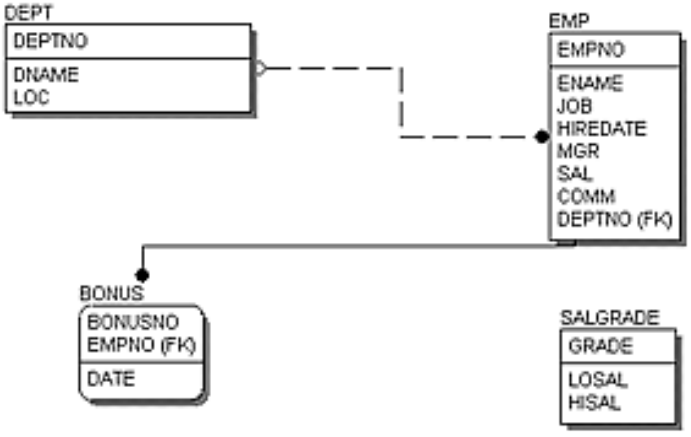


Figura 5.2 - Mostra o modelo de Dados Físico.

A seguir, todos os detalhes dos atributos que serão vistos nos Quadros 5.1 a 5.4.

Tabela: DEPT			
Descrição: Utilizado para armazenar os departamentos			
Atributo	**Tipo do Oracle**	**Tipos do PostgreSQL**	**Descrição**
DEPTNO(PK)	NUMBER(2)	NUMBER(2)	Código de DEPT
DNAME	VARCHAR2(12)	VARCHAR(12)	Nome do departamento
LOC	VARCHAR2(12)	VARCHAR(12)	Localização do departamento

Quadro 5.1 - Tabela DEPT

Tabelas: EMP			
Descrição: Utilizado para armazenar os departamentos			
Atributo	**Tipo**	**Tipos do PostgreSQL**	**Descrição**
EMPNO(PK)	NUMBER(4)	NUMBER(4)	Código de EMP
ENAME	VARCHAR2(10)	VARCHAR(10)	Nome do empregado
JOB	VARCHAR2(9)	VARCHAR(9)	Cargo do empregado
HIRADEDATE	DATE	DATE	Data de inicio
MGR	NUMBER(4)	NUMBER(4)	Gerente do funcionário
SAL	NUMBER(7,2)	NUMBER(7,2)	Salário do empregado
COMM	NUMBER(7,2)	NUMBER(7,2)	Comissão do empregado
DEPTNO(FK)	NUMBER(2)	NUMBER(2)	Código da chave estrangeira

Quadro 5.2 - Tabela EMP

Tabelas: BONUS			
Descrição: Utilizado para armazenar os bônus do funcionário			
Atributo	**Tipo do Oracle**	**Tipos do PostgreSQL**	**Descrição**
BONUSNO(PK)	NUMBER(3)	NUMBER(3)	Código de BONUS
EMPNO(FK)	NUMBER(4)	NUMBER(4)	Chave estrangeira
DATA	DATE	DATE	Data do bônus

Quadro 5.3 - Tabela BONUS

Tabelas: SALGRADE			
Descrição: Utilizado para armazenar os departamentos			
Atributo	**Tipo do Oracle**	**Tipos do PostgreSQL**	**Descrição**
GRADE(PK)	NUMBER(4)	NUMBER(4)	Código do SALGRADE
LOSAL	NUMBER(7,2)	NUMBER(7,2)	Menor faixa salarial
HISAL	NUMBER(7,2)	NUMBER(7,2)	Maior faixa salarial

Quadro 5.4 - Tabela SALGRADE

5.1 Resultado Obtido do Conversor

A obtenção dos resultados se baseou nos modelos de dados acima citados, que ao se conectarem em um banco de dados Oracle, possibilita extrair os nomes das tabelas e seus respectivos atributos (nome dos campos e os tipos de dados). Esses dados serão mostrados em um “textarea” e ao se clicar no “Converter” a conversão será efetuada.

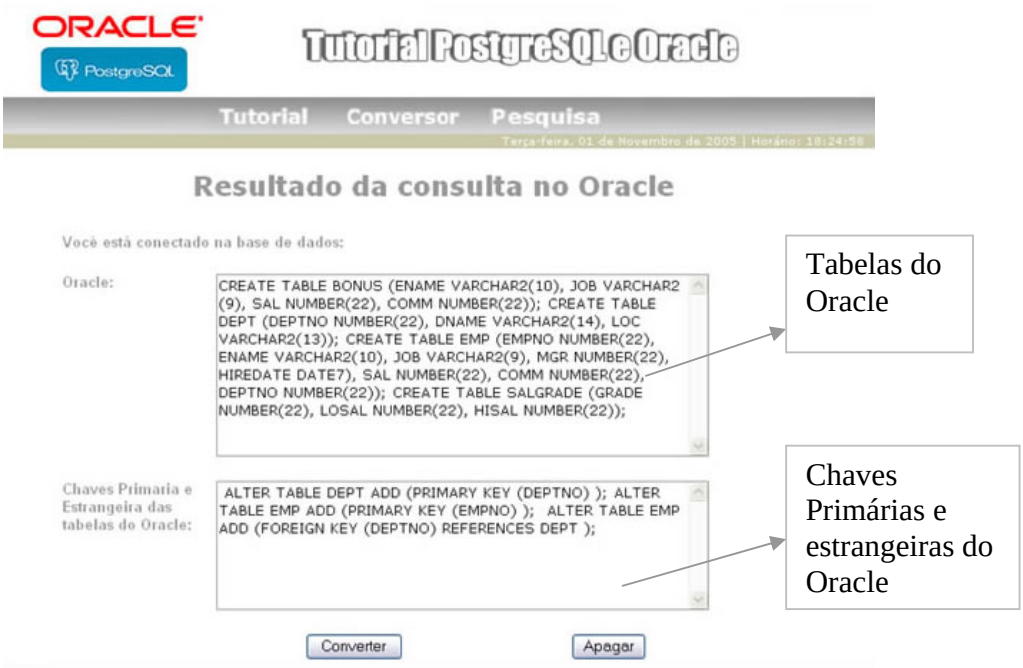


Figura 5.3 - Mostra os dados extraídos do Oracle.

Na Figura 5.4 será mostrada o resultado do conversor de dados do oracle para o PostegreSQL.

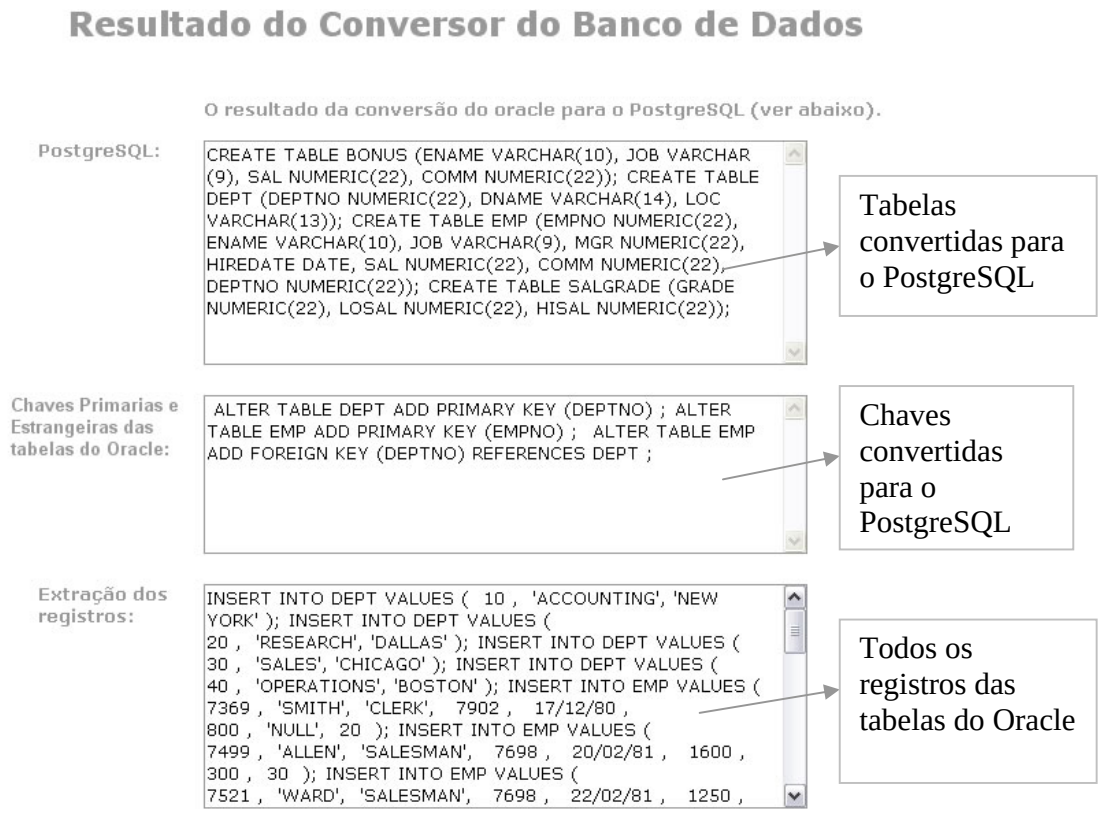


Figura 5.4 - Mostra o Resultado da conversão do Oracle para o PostgreSQL.

Os registros são extraídos das tabelas contidas na base de dados do Oracle. Estes registros estão devidamente formatados para ser executado em um Banco de Dados PostgreSQL.

É importante ressaltar que o banco de dados Oracle e PostgreSQL deverão estar devidamente configurados e instalados para que possa ser utilizado. Na Figura 5.3 são mostrados os dados que foram extraídos do Oracle.

Também foi desenvolvido a conversão de instruções SQL do Oracle para o PostgreSQL. O usuário deverá digitar o comando em um "*textarea*", e ao clicar no botão "converter", será mostrado o resultado da conversão. Na Figura 5.5 é mostrada uma tela do conversor de comandos SQL.



Figura 5.5 - Apresenta o conversor de comandos SQL

No Quadro 5.5 é apresentado o código mostrando como foi feita a conversão dos tipos de dados e comandos SQL. O método utilizado foi uma comparação de *array*, que ao encontrar uma palavra “VARCHAR2” trocará por “VARCHAR”.

```
// Pega a variável pelo método POST
$conversao = $_POST["conver_ora"];
// Array para a conversão dos atributos
$tipo_oracle = array("/CHAR/","/VARCHAR2/","/LONG/","/CLOB/",
"/BLOB/", "/NUMBER/", "/FLOAT/", "/DOUBLE PRECISION/",
"/RAW/","/LONG  RAW/","/DATE/", “/TIMESTAMP/");
$tipo_postgre = array("CHAR","VARCHAR","TEXT", "TEXT",
"BYTEA", "NUMERIC", "NUMERIC", "DOUBLE PRECISION",
"BYTEA","BYTEA", "DATE", "TIMESTAMP");
// Converte as string para Maiúsculas
$data = strtoupper($conversao);
// faz a conversão dos atributos
$data = preg_replace($tipo_oracle, $tipo_postgre, $data);
```

Quadro 5.5 - Parte do código de conversão de tipos

O comando a ser utilizado para fazer a conversão será ilustrado no Quadro 5.6.

```
CREATE TABLE EMP (  empno NUMBER(4) NOT NULL,
                    ename VARCHAR2(10) NOT NULL,
                    job VARCHAR2(9) NULL,
                    hiredate DATE NOT NULL,
                    sal  NUMBER(7,2) NULL,
                    comm  NUMBER(7,2) NULL,
                    mgr  NUMBER(4) NULL,
                    deptno  NUMBER(2) NOT NULL
);
```

Quadro 5.6 - Mostra a instrução SQL do Oracle

No Quadro 5.7, apresenta-se o resultado da conversão de criação de uma tabela.

```
CREATE TABLE EMP (  EMPNO NUMERIC(4) NOT NULL,
                    ENAME VARCHAR(10) NOT NULL,
                    JOB VARCHAR(9) NULL,
                    HIREDATE DATE NOT NULL,
                    SAL  NUMERIC(7,2) NULL,
                    COMM  NUMERIC(7,2) NULL,
                    MGR  NUMERIC(4) NULL,
                    DEPTNO  NUMERIC(2) NOT NULL
);
```

Quadro 5.7 - Mostra o resultado da conversão SQL

5.2 Resultados Obtidos do Tutorial e da Pesquisa

Nesta implementação foram utilizados HTML, DHTML, PHP, javascript e um banco de dados (MYSQL) para inserção, alteração ou modificação nos campos que serão mostrados no navegador de Internet.

A tela principal da página possui uma definição do trabalho, que está ilustrado na Figura 5.6.

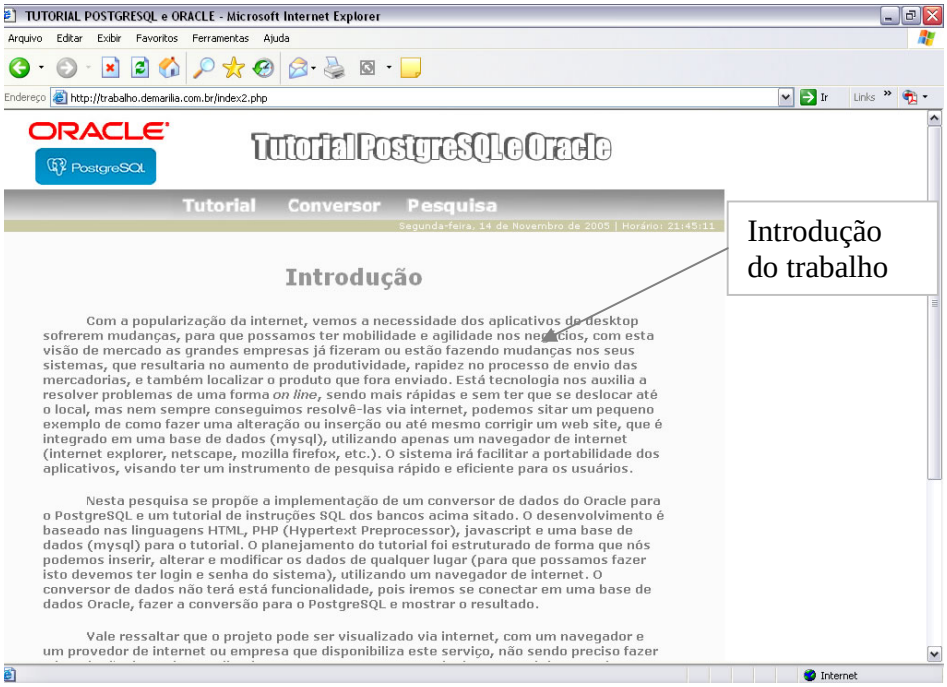


Figura 5.6 - Mostra a página principal

A página do tutorial foi desenvolvida para que o usuário possa fazer as alterações, inserções e modificações *online* dos atributos, porém para isso deverá ter o *login* e senha do administrador. Será mostrada uma ilustração de inserção de dados na Figura 5.7.



Figura 5.7 - Mostra os campos do tutorial

Quando o usuário clicar em “EDITAR E EXCLUIR”, será direcionado para a página ilustrada na Figura 5.8.



Figura 5.8 - Mostra o editor de registros

Para facilitar uma procura no tutorial, foi implementado um sistema de busca de palavras chaves na base de dados. Ilustra-se o resultado de uma consulta utilizando a palavra “**hist**” para fazer a pesquisa. Na Figura 5.9 é apresentado o resultado.

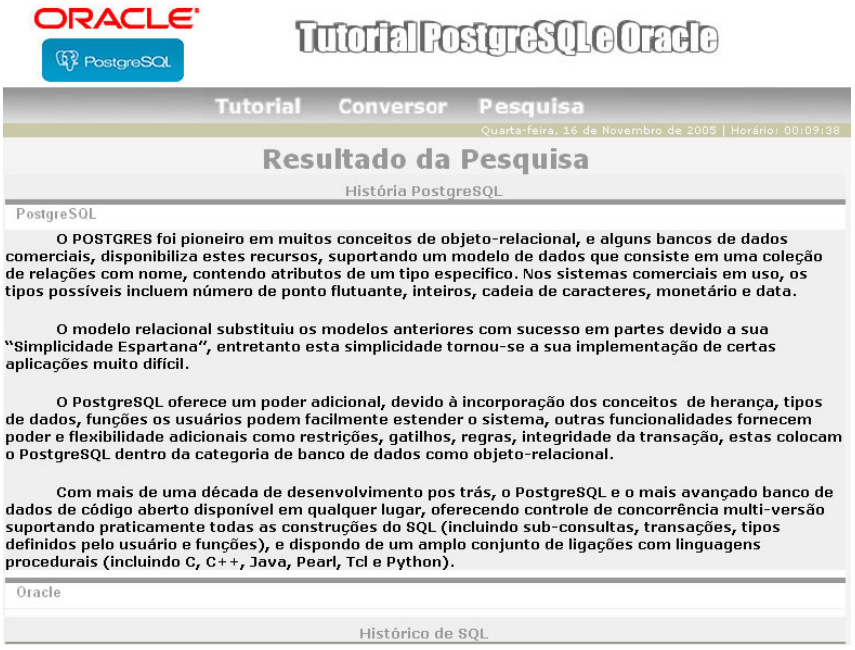


Figura 5.9 - Mostra o resultado da pesquisa.

6. CONCLUSÃO E TRABALHOS FUTUROS

Este Capítulo mostrará a finalização deste trabalho com a conclusão e possíveis trabalhos futuros que poderão ser realizados.

6.1 Conversor de Banco de dados Oracle para PostgreSQL

Para a realização do trabalho foi estudado alguns dos comandos SQL existentes no banco de dados Oracle e PostgreSQL, detalhados no Capítulo 4 desta monografia.

Durante o processo de aprendizagem, a criação do modelo de dados do Oracle e a conversão para o PostgreSQL, foram vistos que os bancos possuem características como objeto-relacional.

Uma das vantagens dos bancos de dados PostgreSQL é possuir código fonte livre, ou seja, não é preciso adquirir uma licença para sua utilização. Outra vantagem é o fato de ter a presença de *stored procedures, triggers*, integridade referencial, etc.

Conclui-se, portando, que os objetivos citados no início desta pesquisa foram alcançados e, podendo ser utilizado para fazer a conversão dos dados em uma base de dados Oracle.

6.2 Trabalhos Futuros

Como proposta para trabalhos futuros, poderá ser explorada a característica do objeto-relacional, conversão de *triggers e stored procedures,* e a criação de uma implementação de rotinas para a inserir os dados obtidos diretamente no SGBDs PostgreSQL de forma automática. Poderia, também, fazer testes nos bancos, para ver o comportamento em uma base de dados grande (com muitas tabelas e dados).

Os tipos de dados poderiam ser implementados em uma base de dados, para que se possa fazer a conversão de qualquer par de SGBDs, desde que estejam cadastrados na metabase de dados.

A concretização destes trabalhos poderiam ajudar numa mudança na base de dados do Oracle para o PostgreSQL, bem como, ajudar a escolher qual banco de dados utilizar nas aplicações de grande, médio e pequeno porte.

REFERÊNCIAS BIBLIOGRÁFICAS

YONG, Chu Shao. **Banco de dados : organização , sistemas e administração.** São Paulo: Atlas, 1990.

Comparativo (2003). Disponível em: <http://www.dbexperts.com.br/documentos/comparativo>. Acesso em 23 de novembro de 2005.

Documentação do PostgreSQL 8.0.0. Disponível em: <http://pgdocptbr.sourceforge.net/>. Acesso em 20 de julho de 2005.

FERNANDES, L.. **Oracle9i Para Desenvolvedores - Oracle Developer 6i Curso Completo.** Axcel Books, 2002.

FOLHA Online (2002). Veja entrevista com o guru de software livre Richard Stallman. Disponível em: <http://www1.folha.uol.com.br/folha/informatica/ult124u10884.shtml>. Acesso em 14 de junho de 2005.

Instalação do PostgreSQL 8.0. Disponível em: <http://www.imasters.com.br/artigo.php?cc=23>. Acesso em 14 de março de 2005.

MANUAL 7.3.4. **Manual de Referência do PostgreSQL 7.3.4.** Disponível em: <http://www.postgresql.org.br/referencia/>. Acesso em 28 de março de 2005.

MANUAL do PHP. Disponível em: <http://www.php.net/manual/pt_BR/index.php>. Acesso em 21 de novembro de 2005.

MELO, R. N.; SILVA, S. D.; TANAKA, Asterio K. **Banco de Dados em Aplicações Cliente/Servidor.** Rio de Janeiro: Infobook, 1997.

MICROSOFT. **Microsoft Ajuda e Suporte.** Converter um Volume FAT16 ou FAT32 para NTFS no Windows XP. Disponível em: <http://support.microsoft.com/default.aspx?scid=kb;pt-br;307881>. Acesso em 15 de março de 2005.

MORELLI, E. T.. **Oracle 9i fundamental : SQL, PL/ SQL e administração.** São Paulo: Érica, 2002.

ORACLE. **Oracle9i SQL Reference Release 2 (9.2).** Disponível em: <http://oraclesvca2.oracle.com/docs/cd/B10501_01/server.920/a96540/toc.htm>. Acesso em 13 de setembro de 2005.

Portal Software Livre. O que é? (2004). Disponível em: <http://portal.softwarelivre.org/whatisit.php>. Acesso em 14 de junho de 2005.

POSTGRESQL 8.0.3.. **PostgreSQL 8.0.3 Documentation.** Disponível em: <http://www.postgresql.org/docs/manuals/>. Acesso em 9 de junho de 2005.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S.. **Sistema de banco de dados.** 3ª ed. São Paulo: Makron Books, 1999.

SOARES, Walace. **PHP 5: conceitos, programação e integração com banco de dados.** São Paulo: Érica, 2004.

THOMSON, L.; WELLING, L.. **PHP e MySQL : desenvolvimento web.** Rio de Janeiro: Campus, 2005.

ANEXO A - Instalação do PostgreSQL

Conforme documentação do PostgreSQL 8.0.0. (2005), o requisito de hardware para a máquina servidora, é ter no mínimo de 8 Mb de espaço livre na memória RAM, será necessário 25 Mb de espaço livre em disco para a instalação mínima, e 100 Mb para a instalação completa de todos os módulos e aplicativos opcionais do PostgreSQL.

Na máquina cliente deve-se ter no mínimo 4 Mb de espaço livre na memória RAM, ter 2 Mb livres no disco, para clientes ODBC, 8 Mb livres para os aplicativos de rede (Protocolos TCP/IP).

Para fazer a instalação precisa-se fazer o download do PostgreSQL 8.0.2, em

http://www.postgresql.org/download o arquivo tem 17.7 MB, o PostgreSQL e suportado por algumas plataformas e versões como Windows 2000, XP e 2003, todos os sistemas unix e suas variações, não suportando as plataformas windows 95, 98 e ME, a menos que instale um emulador de unix nestas plataformas (Cygwin) que pode ser adquirido em http://www.cygwin.com. O sistema de arquivos tem que estar em NTFS para minimizar a perda de dados e eventos de uma grande falha ou de um problema similar, o PostgreSQL não poderá ser instado no sistema FAT, pois não pode fornecer uma ou outra proteção contra corrupção ou algumas características de segurança. Se o sistema operacional for windows XP e estiver no formato FAT terá que fazer a conversão do sistema para NTFS, portanto pode ser utilizar um comando de console do DOS, que ira fazer esta conversão:

CONVERT volume /FS:NTFS

Exemplo CONVERT C: /FS:NTFS

Mais informações http://support.microsoft.com/default.aspx?scid=kb;pt-br;307881

Instalando o PostgreSQL:

1. Descompacte os arquivos **postgresql-8.0.msi** e **postgresql-8.0-int.msi** e execute **postgresql-8.0.msi**, se tiver uma versão anterior a 8.0.2, podemos fazer uma atualização utilizando ”upgrade.bat” que vem junto com o pacote de instalação do PostgreSQL. Quando

iniciar a instalação abrirá uma janela para selecionar o idioma, para ser usado durante a instalação.



Figura A.1 - Mostra os idiomas para a instalação.

2. Aparecerá uma outra tela para fechar todos aplicativos que estiverem abertos, em seguida uma outra tela de informações de instalação.

3. Escolher as opções de configuração para a instalação, os módulos de ”Data Base Serve” e ”User Interface” deverão ser habilitados.

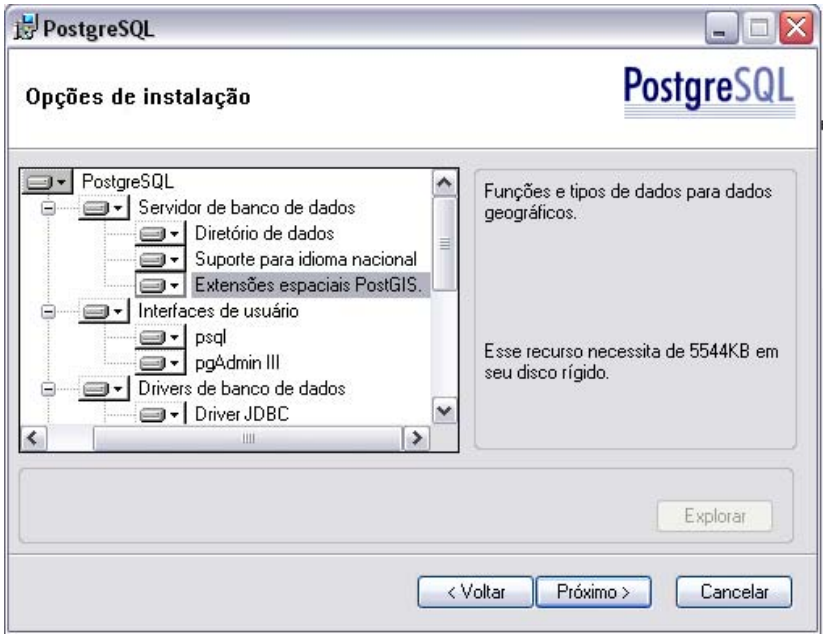


Figura A.2 – Mostra opções de Instalação do PostgreSQL.

4. Terá que optar em instalar o PostgreSQl como um seviço do Windows ou não, se optar na instalação como seviço, deve-se criar um usuário no windows com senha e não ter o privilégio de administrador, esta conta e que será utilizado para o serviço do SGBD, esta configuração pode ser feito pelo PostgreSQL, então você terá que colocar o nome da conta e uma senha, se você deixar que o PostgreSQL coloque uma senha, está senha será gerada e dificilmente você conseguira memorizar, como mostra na Figura A.3.



Figura A.3 – Mostra senha gerada pelo PostgreSQL.

5. Inicializar o agrupamento do BD, nesta janela você terá que se configurar em qual porta o sevidor irá rodar, e se optar em aceitar coneções em todas os endereços, e não apenas localhost, você ainda terá que conceder acesso aos endereços de servidores, ou de uma rede especifica, e configurar o arquivo ”pg_hba.conf”, terá que selecionar o local (locale), a codificação e o super usuário da conta interna do SGBD, por motivo de segurança não coloque a mesma senha da conta de serviço.

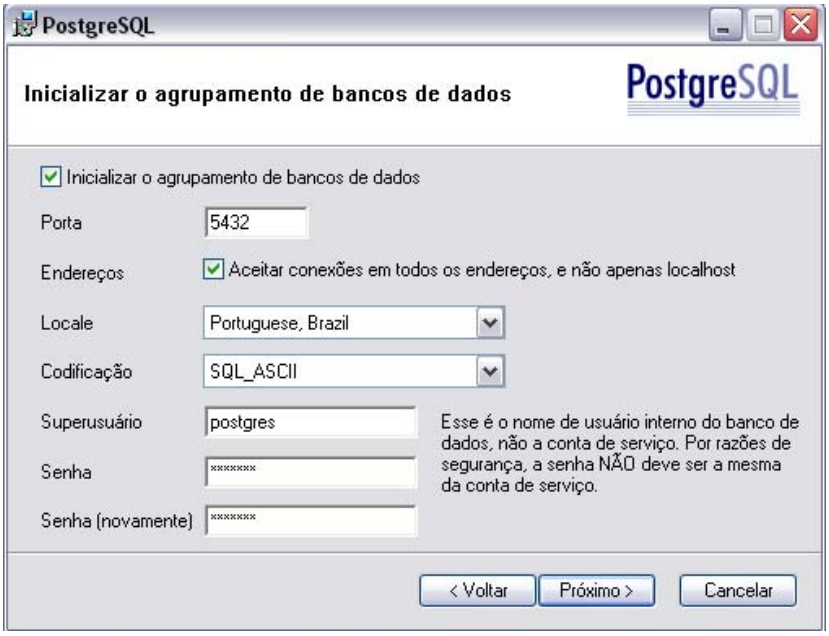


Figura A.4 – Mostra a inicialização do Agrupamento de Banco de Dados.

6. Nesta outra janela, terá que escolher os módulos adicionais a serem instalados.

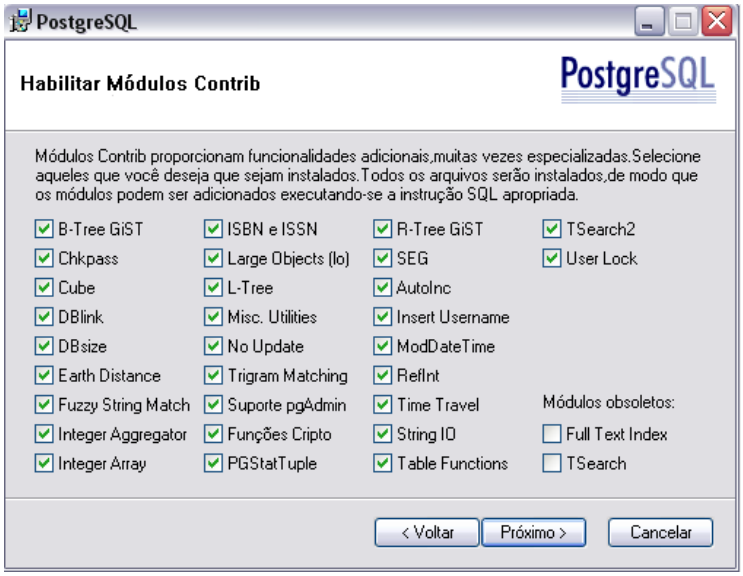


Figura A.5 – Mostra os módulos contribuintes.

Função de cada módulo contribuinte:

Tabela A.1 – Mostra funções de instalação do PostgreSQL.

B-Tree	Uma implementação do B-Tree usando GiST para vários tipos de dados
ChkPass	Tipo de senha que é automaticamente checada e convertida na entrada, sendo armazenada de forma criptografada
Cube	Tipo de dados que representam cubos multidimensionais
DBLInk	Funções que retornam resultados a partir de um banco de dados remoto.
DBSize	Módulo utilizado no PgAdmin que contém funções para a busca de tamanho de um objeto no banco de dados.
Earth Distance	Módulo que contém duas diferentes abordagens para calcular grandes distâncias levando em consideração a curvatura da Terra. Para instalar esse módulo você necessita instalar também o módulo Cube.
Fuzzy String Match	Funções para comparação "fuzzy" de strings.
Integer Aggregator	Um integrador/enumerados de inteiros.
Integer Array	Essa é uma da estrutura de dados RD-tree utilizando a interface GiST para PostgreSQL.
ISBN e ISSN	Tipos de dados para ISBN (livros) e ISSN (seriais)
Large Objects (lo)	Extensão do PostgreSQL para tratar Large Objects.
L-Tree	Uma implementação de tipos de dados, métodos de acesso indexados e consultas para dados organizados em uma estrutura de árvore.
Misc. utilities	Diversas funções úteis para o PostgreSQL.

Tabela A.1 – Mostra funções de instalação do PostgreSQL. (Continuação)	
No Update	Módulo que implementa um gatilho para prevenir atualizações em colunas isoladas.
Trigram Matching	Módulo que proporciona funções e classes de índices para determinar a similaridade de textos baseados em comparação de trigramas.
Suporte PgAdmin	Conjunto de funções empregadas para proporcionar funcionalidades adicionais no PgAdmin.
Funções Cripto	Funções criptográficas para o PostgreSQL.
PGStartTuple	Retorna o comprimento da tabela, a percentagem de tuplas "mortas" em uma tabela e outras informações.
R-Tree Gist	Uma implementação de R-Tree empregando GiST.
SEG	Um tipo definido pelo usuário representando medidas de laboratório, tais como intervalo de de ponto flutuante.
String IO	Módulo que implementa funções de saída que podem ser empregadas no lugar das padrões, de modo a obter o valor de campos de texto impressos no formato utilizado por strings C.
Table Functions	Uma amostra de funções C que retornam setof e setof composto.
TSearch2	Uma extensão de busca de texto completa para PostgreSQL. Esse módulo deve ser empregado preferencialmente em relação ao "Full Text Index" e "TSearch" para novas aplicações.
User Lock	Esse módulo proporciona suporte para bloqueio cooperativo de longo prazo a nível de usuário.
Full Text Index	Uma tentativa de algum tipo de Indexação de Texto Completo para PostgreSQL. O módulo TSearch2 é recomendados em seu lugar, para novas aplicações.
TSearch	Uma extensão de busca de texto completo para PostgreSQL. O módulo TSearch2 é recomendados em seu lugar, para novas aplicações.

7. Se estiver instalado o modulo PostGIS, abrirá uma janela perguntando se deseja habilitar a função, e carregar dados de referência para o Banco de Dados template1, ou poderá habilitar mais tarde em BD específicos.

8. Depois da instalação aparecerá uma janela, para assinar a lista de e-mail jgsql-announce, para receber informações de atualizações, correção de possíveis erros e novas versões.

Anexo B - Instalação do Oracle

O requisito de hardware necessário para a máquina servidora é um PC baseado em processador Pentium 166 MHz (recomenda-se 233 MHz), com RAM de 128 MB (recomenda-se 512 MB), disco configurado com NTFS deve-se ter 2,8 GB mais 275 MB livre no disco para a base de dados, e com o disco configurado com FAT32 devemos ter 4,5 GB de espaço livre em disco e mais 140 MB. No disco deve-se ter 400 MB livres apontando para a variável de ambiente TEMP, Para a área de paginação (*swap*) deve-se ter no mínimo de 200 MB.

Para a máquina cliente, devemos instalar em um PC baseado em Pentium 166 MHz ou superior com no mínimo 64 MB de RAM (recomenda-se 128 MB), espaço em disco de no mínimo 800 MB de espaço livre.

Instalando o Oracle:

1. Nesta primeira janela de instalação vemos que se pode desinstalar produtos do Oracle, ver o que foi instalado.

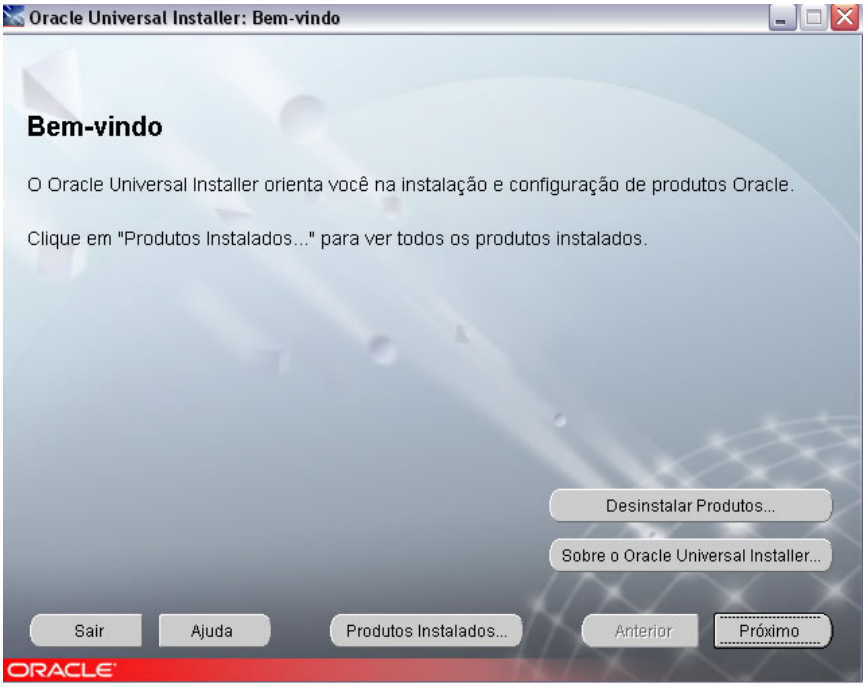


Figura B.1 – Mostra o início da instalação do Oracle.

2. Ao ser pressionado o “Próximo”, deve-se indicar o caminho dos módulos para instalar (origem), o nome e onde serão instalados e deverão estar indicados nos devidos campos (destino).

3. Depois de ser configurado o item anterior, irá ser escolhido o produto a ser instalado, neste que foi escolhido o primeiro item (Oracle9i Database 9.2.0.1.0), também se pode escolher os idiomas desta instalação.

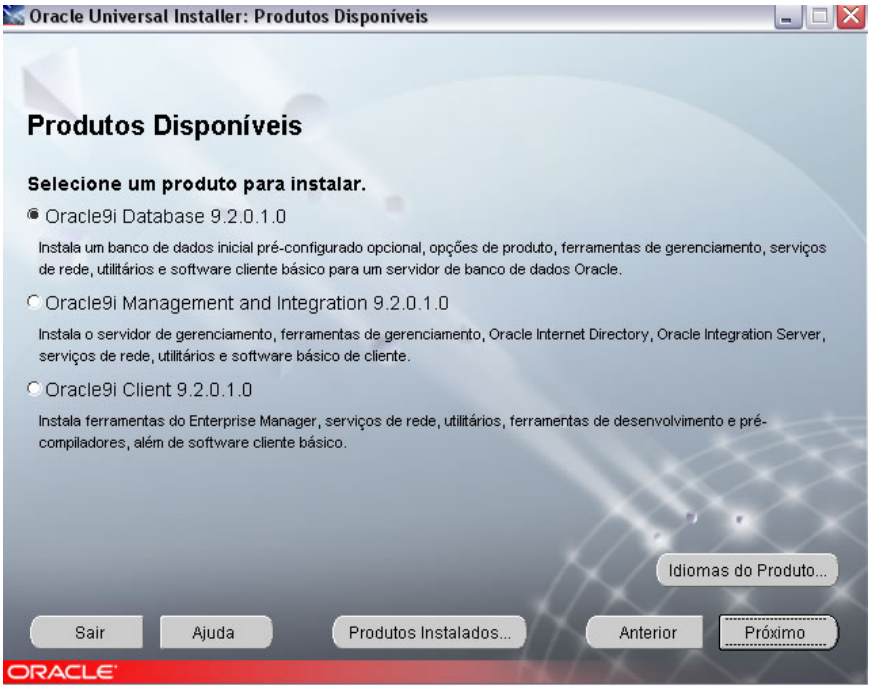


Figura B.2 – Mostra os produtos disponíveis para a instalação.

4. A configuração desta parte consistem em criar o banco de dados global que deverá ser da seguinte forma “nome.dominio”, neste utilizamos “teste.localhost”. O SID deverá ser informado, o sistema irá sugerir um, podendo mudar, o banco de dados e referenciado por pelo menos uma instância pelo Oracle, identificada exclusivamente a qualquer instância por um SID neste computador.

5. Agora se deve escolher uma das quatro opções de instalação, e será mostrado o espaço de utilização em cada instalação.

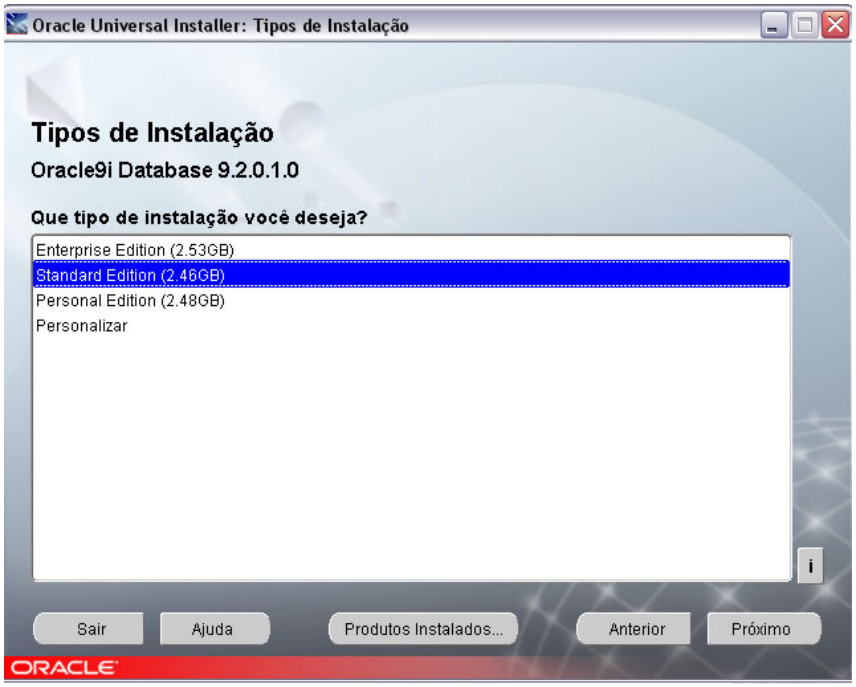


Figura B.3 – Mostra os tipos de instalação.

6. Próxima tela mostra cinco opções de escolha para a configuração do bando de dados: finalidade geral, processamento de transações, data warehouse, personalizada, somente software, bem como a explicação de cada item, ilustrada na Figura B.4.

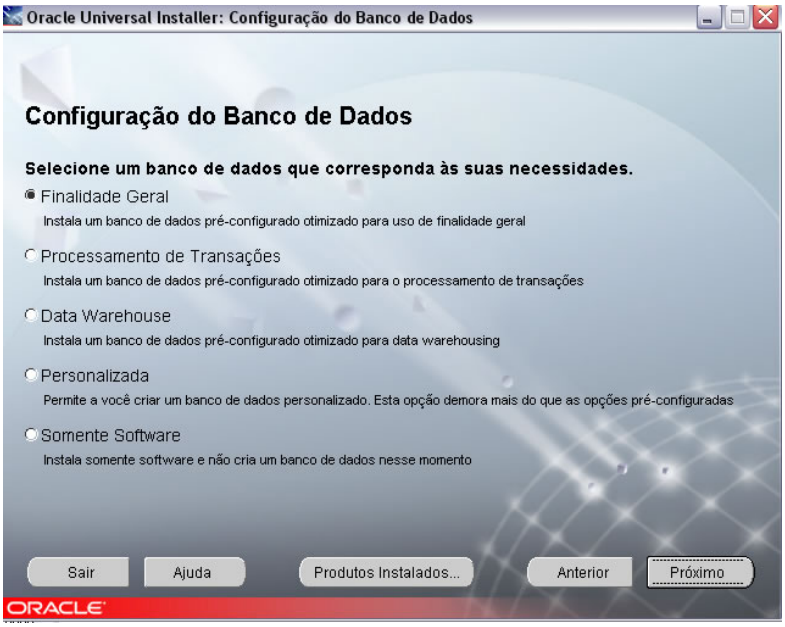


Figura B.4 – Mostra a configuração do Banco de Dados.

7. Escolher a porta em que o Oracle MTS (Microsoft Transaction Services) Recovery Services atenderá a solicitação neste computador.

8. Deve-se indicar onde deverá ser instalado o arquivo do bando de dados, e recomenda-se que o arquivo de banco e o software Oracle deverá ser instalados em discos diferentes, para melhor organização e otimização do banco.

9. A escolha do conjunto de caracteres determinará qual será utilizado no banco de dados, termos três opções de configuração, nesta, usamos o conjunto de caracteres default.

10. Para a finalização da instalação, será mostrado o resumo dos componentes que serão instalados, e então começará a instalação.

Anexo C - Principais Comandos SQL do PostgreSQL

Segundo a documentação do PostgreSQL 8.0.0 (2005), os comandos em SQL são formados por uma seqüência de termos (Tokens), terminados por ponto-e-vírgula (“;”). Um termo pode ser uma palavra chave, um identificador entre aspas, um literal ou constante, ou um caractere especial. Os padrões suportados são ANSI SQL-89, SQL-92 e SQL-99.

Exemplo:

```
SELECT * FROM TESTE;
UPDATE TESTE SET  A= 1;
INSERT INTO TESTE VALUES (1, 'TESTE');
```

Quadro C.1 - Mostra exemplos de comandos SQL do PostgreSQL

As palavras chave neste comando são SELECT, UPDATE, INSERT, e os identificadores são TESTE, A.

Os tipos de dados suportados no PostgreSQL são listados na Tabela C.1

Tabela C.1 - Tipos de dados suportados no PostreSQL.

Nome	Tamanho	Faixa/Precisão	Descrição
Smallint	2 bytes	-32768 à +32767	precisão fixa pequena com faixa
Integer	4 bytes	-2147483648 à +2147483647	escolha usual precisão fixa para
Bigint	8 bytes	-9223372036854775808 à 9223372036854775807	precisão fixa grande com faixa
Decimal	variável	sem limite	precisão especificada pelo usuário, exato
Numeric	variável	sem limite	precisão especificada pelo usuário, exato
Real	4 bytes	precisão de 6 dígitos decimais	precisão inexato variável,
Double precision	8 bytes	precisão de 15 dígitos decimais	precisão inexato variável,
Serial	4 bytes	1 a 2147483647	inteiro autoincremental

Tabela C.2 - Tipos de dados suportados no PostreSQL(continuação)			
Nome	**Nome**	**Nome**	**Nome**
Bigserial	8 bytes	1 a 9223372036854775807	inteiro grande autoincremental
Money	4 bytes	-21474836.48 a +21474836.47	quantidade de dinheiro
Character , Varying(n), Varchar(n)	-	-	Comprimento variável com limite
Character(n), Char(n)	-	-	comprimento fixo, completado com brancos
Text	-	-	comprimento não limitado variável
Bytea	-	4 bytes mais a cadeia binária	cadeia binária de comprimento variável
Timestamp [(p)] [without time zone]	8 bytes	4713 AC a 5874897 DC	tanto data quanto hora
Timestamp [(p)] with time zone	8 bytes	4713 BC a 5874897 DC	tanto data quanto hora, com zona horária
Interval [(p)]	12 bytes	-178000000 anos a 178000000 anos	intervalos de tempo
Date	4 bytes	4713 AC a 32767 DC	somente datas
Time [(p)] [without time zone]	8 bytes	00:00:00.00 a -23:59:59.99	somente a hora do dia
Time [(p)] with time zone	12 bytes	00:00:00.00+12 a -23:59:59.99-12	somente a hora do dia, com zona horária
Boolean	-	‘TRUE’, ‘t’, ‘true’, ‘y’, ‘yes’, ‘1’ ou ‘FALSE’, ‘f’’ , ‘false’, ‘n’, ‘no’, ‘0’	-

Para criar uma Base de Dados, utiliza-se o comando, descrito no Quadro C.2.

```
CREATE DATABASE nome_db
[ [ WITH ] [ OWNER [=] dono_do_banco_de_dados ]
[ TEMPLATE [=] modelo ]
[ ENCODING [=] codificação ]
[ TABLESPACE [=] espaço_de_tabelas ] ] ;
```

Quadro C.2 - Mostra a criação de um banco de dados

Com a criação do banco, aparecerá uma mensagem CREATE DATABASE, isto significa que o banco foi criado.

Usualmente usam-se letras maiúsculas para palavras chaves e letras minúsculas para nomes. Existe um segundo tipo de identificador: o identificador delimitado ou identificador entre aspas (“item1”). O identificador delimitado e sempre um identificador é não uma palavra chave. Portando uma palavra chave “*create*” pode ser usado para fazer referência a uma outra tabela ou coluna chamada “*create*”, e sem as aspas e sempre uma palavra chave. Colocar identificadores entre aspas torna-se diferentes as letras maiúsculas e minúsculas, enquanto os nomes envoltos de aspas serão convertidos para letras minúsculas. Exemplo: nome, NOME, “nome” são considerados os mesmos pelo PostgreSQL, enquanto que “NOME” e “nome” são diferentes, pois são diferentes entre os três e entre si.

A criação das tabelas pode ser da seguinte maneira, como descrito no Quadro C.3

```
CREATE TABLE nome_tabela
(item1 Integer PRIMARY KEY,
item2 varchar(20),
item_outra_tabela varchar(20) REFERENCES
nome_da_tabela_refenciada (item_outra_tabela),....);
```

Quadro C.3 - Mostra a criação de tabelas com chaves

A palavra reservada PRIMARY KEY é uma restrição de chave, que é uma combinação de UNIQUE com NOT NULL. A chave primária identifica uma coluna ou um grupo de colunas, para se tornar um identificador único das linhas da tabela.

A palavra reservada REFERENCES significa que o campo citado da tabela acima será referenciado pela tabela “nome_da_tabela_refenciada”, está referência é denominado de FOREIGN KEY ou chave estrangeira. Esta chave pode restringir colunas ou grupo de colunas, como mostra a sintaxe apresentada no Quadro C.4.

```
CREATE TABLE nome_tabela (item1 Integer PRIMARY KEY,
item2 Varchar(20), item3 Varchar(20),
FOREIGN KEY (item2, item3)
REFERENCES nome_da_tabela_refenciada
(item2_outra_tabela, item3_outra_tabela));
```

Quadro C.4 - Mostra a criação de chaves estrangeiras

Para removendo uma tabela, é necessário seguir a sintaxe apresentada a seguir:

```
DROP TABLE nome_tabela;
```

Deve-se tomar cuidado com este comando, pois ele removerá a tabela, e o que estiver inserido na mesma.

Para inserir linhas em uma tabela, é necessário seguir a sintaxe apresentada no Quadro C.5:

```
INSERT INTO nome_tabela VALUES (‘item1’, ‘item2’, ...);
ou
INSERT INTO nome_tabela (nome_campo1, nome_campo2, ...)
VALUES (‘item1’, ‘item2’, ...);
```

Quadro C.5 - Mostra o comando para inserir registro na tabela

As constantes que não são numéricas, geralmente estão entre apóstrofos (‘). Neste primeiro exemplo de inserção de linhas os campos deverão estar devidamente na mesma ordem da criação das tabelas. No segundo exemplo os campos deverão estar na mesma ordem (nome_campo1 inserirá o item1).

Tem-se também uma outra forma de inserir grandes dados em uma tabela, o comando é COPY.

```
COPY nome_tabela FROM ‘/caminho_arquivo/nome_arquivo.txt’;
```

Este, geralmente é mais rápido, porém o comando INSERT tem mais flexibilidade. O arquivo deverá ser acessado pelo servidor e não pelo cliente, porque o servidor lê o arquivo diretamente.

Para alterando uma tabela, é necessário seguir a sintaxe apresentada a seguir:

```
ALTER TABLE nome_tabela FUNÇÃO nome_campo tipo_dado;
```

É permitido fazer diversas alterações:

Para alterar a tabela inserindo campos, é necessário seguir a sintaxe apresentada a seguir:

```
ALTER TABLE nome_tabela ADD COLUMN nome_campo
tipo_dado;
```

Pode-se acrescentar uma restrição para uma coluna no exemplo a seguir:

```
ALTER TABLE nome_tabela ADD COLUMN nome_campo tipo_dado
CHECK (nome_campo <> ‘’);
```

Para alterar a tabela removendo campos, é necessário seguir a sintaxe apresentada a seguir:

```
ALTER TABLE nome_tabela DROP COLUMN nome_campo;
```

Para alterar a tabela inserindo restrições, é necessário seguir a sintaxe apresentada a seguir:

```
ALTER TABLE nome_tabela ADD CHECK (nome_campo <> ‘’);
```

O PostgreSQL permite inserir restrições de um nome em uma tabela, o comando a seguir mostra um exemplo:

```
ALTER TABLE nome_tabela ADD CONSTRAINT inq_id_nome
UNIQUE (id_nome);
```

Neste comando, utilizar a palavra chave CONSTRAINT e o nome da restrição (inq_id_nome). A palavra chave UNIQUE, garante a atomicidade de que os dados da coluna ou do grupo de colunas sejam únicos entre todas as linhas das colunas.

Pode-se inserir uma chave primária ou uma chave estrangeira:

```
ALTER TABLE nome_tabela ADD PRIMARY KEY (grade);
```

O comando anterior adiciona uma chave primária em um tabela, os dados serão verificados e devem satisfazer a restrição.

A seguir a inserção de uma chave estrangeira.

```
ALTER TABLE nome_tabela ADD FOREIGN KEY (nome_campo)
REFERENCES tabela_referenciada;
```

Este comando insere uma chave estrangeira na tabela descreve o nome da tabela e referencia a tabela de onde a chave primária está.

Para alterar a tabela retirando restrições, é necessário seguir a sintaxe apresentada a seguir:

```
ALTER TABLE nome_tabela DROP CONSTRAINT
nome_restrição;
```

Pode se retirar a restrição do campo de não ser nulo, como no exemplo a seguir:.

```
ALTER TABLE nome_tabela ALTER COLUMN id_nome
DROP NOT NULL;
```

Para alterar a tabela mudando o valor do campo, é necessário seguir a sintaxe apresentada a seguir:

```
ALTER TABLE nome_tabela ALTER COLUMN nome_campo SET
DEFAULT novo_valor;
```

Também e permitido remover o valor padrão do campo, neste o valor padrão é nulo.

```
ALTER TABLE nome_tabela ALTER COLUMN nome_campo SET
DEFAULT;
```

Para alterar a tabela mudando o nome da coluna, utilizou-se o comando a seguir:

```
ALTER TABLE nome_tabela RENAME COLUMN nome_antigo
TO nome_novo;
```

Para alterar o nome da tabela, utilizou-se o comando a seguir:

```
ALTER TABLE nome_antigo RENAME TO nome_novo;
```

Para se fazer uma consulta de todas as linhas de uma tabela:

Para se fazer uma consulta dos elementos de uma tabela, é necessário seguir a sintaxe apresentada a seguir:

```
SELECT * FROM nome_tabela;
```

Obs: o asterisco (*) significa todas as colunas da tabela.

Pode-se fazer consulta com qualquer expressão arbitrária, como mostra no exemplo a seguir:

```
SELECT nome_tabela, (valor1 + valor2)/2 AS nome_novo, outro_nome
FROM nome_tabela;
```

Esta expressão anterior, lista o conteúdo da tabela e a expressão AS mudam o nome do campo calculado na expressão (valor1 + valor2)/2.

Para se fazer uma consulta com expressões booleanas (AND, OR e NOT), é necessário seguir a sintaxe apresentada a seguir:

```
SELECT * FROM nome_tabela
WHERE nome_campo = ‘valor_campo’
AND nome_campo > 0;
```

A expressão WHERE significa que será mostrado o nome do campo (nome_campo) igual ao valor do campo (valor_campo) a ser procurado nesta consulta, pode-se inserir mais uma clausula, o AND, para filtrar a consulta, verificando as duas expressões para obter a resposta desejada. Será utilizado operador de comparação para selecionar as linhas. Os operadores de comparação são: >,<, >=, <=, <>, !=.

A seguir instruções para a manipulação das consultas: o comando BETWEEN seleciona linha em um limite de números menor e maior.

```
SELECT * FROM nome_tabela WHERE nome_campo BETWEEN
numero_menor AND numero_maior;
```

Temos o comando de negação NOT BETWEEN que retorna as linhas que não estão entre o limite inferior e superior.

Para mostrar uma consulta em ordem alfabética de um campo, a seguir um exemplo:

```
SELECT DISTINCT nome_tabela
FROM nome_tabela ORDEY BY nome_tabela;
```

Com a cláusula DISTINCT eliminam-se as palavras que são iguais.

A consultas com uma junção de duas ou mais tabelas, será mostrada nos exemplos a seguir:

```
SELECT * FROM nome_tabela1, nome_tabela2
WHERE nome_campo1 = nome_campo2;
```

Esta consulta listará apenas o campo das duas tabela que tiverem os mesmos nomes na cláusula WHERE. Se em uma tabela tiver o nome_campo3 e na outra tabela não tiver o mesmo nome, o resultado da resposta não mostra o campo (nome_campo3).

Pode-se escrever de uma outra forma chamada de junção interna (INNER JOIN), mas esta não é muito utilizada, está será mostrado logo abaixo da junção cruzada (CROSS JOIN).

A consultas de junções cruzada CROSS JOIN, será mostrada no exemplo a seguir:

```
SELECT * FROM nome_tabela CROSS JOIN nome_tabela1;
```

A junção cruzada irá juntar todas as colunas da tabela1 com todas as colunas da tabela2, formando uma matriz de N * M linhas.

Os tipos de funções qualificadas são INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN.

A consultas de junções qualificadas INNER JOIN, será mostrada no exemplo a seguir:

```
SELECT * FROM nome_tabela1
INNER JOIN nome_tabela2 ON (nome_tabela1.nome_campo1 =
nome_tabela2.nome_campo2);
```

No campo nome_tabela.nome_campo deve-se colocar o nome da tabela para diferenciar a tabela que irá fazer a consulta.

A cláusula ON lista todas as colunas das tabelas mesmo tendo o mesmo código, e com a cláusula USING mostrará apenas uma vez na tabela de saída, por exemplo, se as duas tabelas tiver três colunas, então mostrara três colunas com os resultados da consulta.

Obs: A cláusula NATURAL e a forma abreviada da USING.

Será mostrado um exemplo do comando USING, a seguir:

```
SELECT * FROM nome_tabela INNER JOIN nome_tabela1 USING
(nome_campo);
```

Um exemplo do comando NATURAL INNER JOIN, será mostrado a seguir:

```
SELECT * FROM nome_tabela NATURAL INNER JOIN nome_tabela1;
```

A consultas de junções externas à esquerda e a direita, será apresentada a seguir:

```
SELECT * FROM nome_tabela1
LEFT OUTER JOIN nome_tabela2 ON
(nome_tabela1.valor_campo1 = mome_tabela2.valor_campo2);
```

Neste caso o nome da tabela que vai à esquerda dos operadores irá listar todo o seu conteúdo uma única vez e o nome da tabela que vai a direita listara apenas os campos que atenderem a cláusula. Ao se listar um campo na esquerda e não tiver nenhum campo que se relacione com o campo da direta, então este campo irá aparecer nulo (vazio). O mesmo se aplica na junção à direita.

```
SELECT * FROM nome_tabela1
RIGHT OUTER JOIN nome_tabela2
ON (nome_tabela1.valor_campo1 = mome_tabela2.valor_campo2);
```

A consulta de junções com FULL OUTER JOIN, será mostrada a seguir:

```
SELECT * FROM nome_tabela FULL JOIN nome_tabela1
ON nome_tabela.valor_campo1 = nome_tabela1.valor_campo2;
```

Para fazer consultas trocando o nome da tabela (“ALIASES”), é necessário seguir a sintaxe apresentada a seguir:

```
SELECT * FROM nome_tabela1 AS tab1, nome_tabela tab2
WHERE tab1.valor_campo1 = tab2.valor_campo2;
```

Obs: No PostreSQL a cláusula “AS” pode ser omitida.

O postgreSQL suporta funções de agregação, calculando um único resultado para várias linhas, algumas dessas são: contar os resultados (count), somar (sum), calcular a média (avg), o valor máximo (max), o valor mínimo para um conjunto de linhas.

```
SELECT max(nome_campo) FROM nome_tabela;
ou
SELECT nome_campo FROM nome_tabela1
WHERE nome_campo = (SELECT max(nome_campo)
FROM nome_tabela);
```

Na consulta anterior, a palavra WHERE está comparando se o elemento(nome_campo) é igual a outra expressão (SELECT max(nome_campo) FROM nome_tabela).

Uma consulta com o GROUP BY será ilustrada a seguir:

```
SELECT nome_campo1, max(nome_campo2)
FROM nome_tabela
GROUP BY nome_campo1
HAVING MAX(nome_campo2) < 40;
```

A instrução anterior está listando o máximo (nome_campo) agrupado pelo grupo (nome_campo), isto significa que não aparecerá nome de campos iguais, com o seu numero máximo, e depois o HAVING verifica a cláusula depois de ter feito o grupo e então mostra o resultado. A cláusula DESC ordena a consulta do maior para o menor e o ASC do menor para o maior.

O exemplo a seguir mostra uma atualização de um registro:

```
UPDATE nome_tabela
SET nome_campo1 = nome_campo1 + 2, nome_campo2 =
nome_campo2 -2 WHERE nome_campo3 > 30;
```

Para validar uma atualização usa-se o COMMIT depois do comando, e quando o resultado não foi o esperado, pode-se retornar com ROLLBACK. Pode-se optar em deixar o COMMIT automático, nas configurações do BD.

No exemplo a seguir mostra a exclusão de um registro:

```
DELETE FROM nome_tabela
WHERE nome_campo = ‘valor_campo’;
```

O comando anterior irá excluir os campos que forem iguais ao nome do valor_campo, escritos após WHERE. Tenha cuidado ao usar este comando, porque ele pode excluir todo o conteúdo da tabela.

```
DELETE FROM nome_tabela;
```

Pode se fazer uma combinação de consultas utilizando as cláusulas UNION, INTERSECT, EXCEPT que será mostrado a seguir:

```
(SELECT * FROM nome_tabela)
UNION ALL
(SELECT * FROM nome_tabela1);
```

O UNION, anexa o resultado da consulta1 com a consulta2 e elimina linhas duplas no resultado, quando se utiliza o ALL não serão eliminadas as linhas que são duplas.

O INTERSECT retorna o resultado das duas consultas, eliminando as linhas duplas, desde que não utilize a cláusula ALL.

O EXCEPT retorna o resultado da consulta1, e que não estiver presente na consulta2, eliminando as linhas duplas, desde que não utilize o ALL.

Os operadores matemáticos são utilizados para realizar cálculos nas consultas ou alterações nos registros, são listados na Tabela C.2:

Tabela C.2 - Mostra operadores matemáticos do PostgreSQL

Operador	Descrição
+	adição
-	subtração
*	multiplicação
/	divisão (divisão inteira trunca o resultado)
%	módulo (resto)
^	exponenciação
|/	raiz quadrada
||/	raiz cúbica
!	fatorial
!!	fatorial (operador de prefixo)
@	valor absoluto
&	AND bit a bit
|	OR bit a bit
#	XOR bit a bit
~	NOT bit a bit
<<	deslocamento à esquerda bit a bit
>>	deslocamento à direita bit a bit

Funções matemáticas do postgreSQL: as funções que utilizam o DOUBLE PRECISION “dp” utilizam as bibliotecas do hospedeiro, a precisão e o comportamento em casos limites podem, portanto, variar dependendo do sistema hospedeiro. A cláusula “FROM nome_tabela” pode ser omitida nas consultas abaixo.

As funções matemáticas: ABS, CEIL, FLOOR, MOD, POWER, ROUND, SIGN, SQRT, TRUNC, serão detalhados no tutorial.

- As cláusula retornarão um valor absoluto de n, como no exemplo abaixo:

SELECT ABS(-10), ABS (nome_campo) FROM nome_tabela;

O resultado será: ABS(-10)

10

As funções e operadores de manipulação de caractere: CHAR_LENGTH ou CHARACTER_LENGTH, LOWER, UPPER, OVERLAY, POSITION, SUBSTRING, TRIM,

BTRIM, INITCAP, LENGTH, LPAD, RPAD, LTRIM, RTRIM, REPEAT, REPLACE, STRPOS, SUBSTR, TRANSLATE, serão detalhados no tutorial.

Um exemplo de manipulação de caractere:

- A cláusula CHAR_LENGTH (‘cadeia_de_caracteres’) ou CHARACTER_LENGTH (‘cadeia_de_caracteres’), retorna a quantidade caractere definido na expressão.

```
SELECT CHAR_LENGTH('José') FROM nome_tabela;
```

O resultado será: CHAR_LENGTH ('José')

4

A cláusula LIKE cadeia_de_caracteres (NOT) LIKE cadeia_de_caracteres [ESCAPE caractere_de_escape], retorna o conjunto de caractere, especificado na expressão. A negação (NOT) da expressão pode ser usada para achar uma expressão que não esta no conjunto de caractere.

```
SELECT nome_campo FROM nome_tabela WHERE nome_campo
LIKE ‘%a\_b%’ ESCAPE ‘\\’;
```

Neste comando utiliza-se “%” para selecionar qualquer cadeia de zeros ou mais caracteres, e caractere “_” representa um único caractere. Para representar os caracteres sublinhados e percentagem, e para não corresponder a outros caracteres, a cláusula LIKE deve vir precedida da palavra ESCAPE ‘\\’. O contra barra é o caractere de escape padrão no PostgreSQL.

Modelos padrão de formatação de datas e horas:

Tabela C.3 - Apresenta uma lista de formatação de datas e horas do PostgreSQL.

Modelo	**Descrição**
HH, HH12	hora do dia (01-12)
HH24	hora do dia (00-23)
MI	minuto (00-59)
SS	segundo (00-59)
MS	milissegundo (000-999)
US	microssegundo (000000-999999)
SSSS	segundos após a meia-noite (0-86399)
AM ou A.M. or PM ou P.M., am ou a.m. or pm ou p.m.	indicador de meridiano
Y,YYY	ano (4 e mais dígitos) com vírgula

Tabela C.3 - Apresenta uma lista de formatação de datas e horas do PostgreSQL. (Continuação)	
Modelo	**Descrição**
YYYY	ano (4 e mais dígitos)
YYY	últimos 3 dígitos do ano
YY	últimos 2 dígitos do ano
Y	último dígito do ano
IYYY	ano ISO (4 ou mais dígitos)
IYY	últimos 3 dígitos do ano ISO
IY	últimos 2 dígitos do ano ISO
I	último dígito do ano ISO
BC ou B.C. or AD ou A.D., bc ou b.c. or ad ou a.d.	indicador de era (maiúscula)
MONTH, Month, month	nome completo do mês (9 caracteres completado com espaços)
MON, Mon, mon	nome abreviado do mês (3 caracteres)
MM	número do mês (01-12)
DAY, Day, day	nome completo do dia(9 caracteres completado com espaços)
DY, Dy, dy	nome abreviado do dia (3 caracteres)
DDD	dia do ano (001-366)
DD	dia do mês (01-31)
D	dia da semana (1-7; Domingo é 1)
W	semana do mês (1-5) onde a primeira semana começa no primeiro dia do mês
WW	número da semana do ano (1-53) onde a primeira semana começa no primeiro dia do ano
W	semana do mês (1-5) onde a primeira semana começa no primeiro dia do mês
CC	século (2 dígitos)
J	Dia Juliano (dias desde 1 de janeiro de 4712 AC)
Q	Trimestre
RM, rm	mês em algarismos romanos (I-XII; I=Janeiro)
TZ, tz	nome da zona horária

Funções de datas e horas no PostgreSQL: AGE, CURRENT_DATE, CURRENT_TIMESTAMP, DATA_PART, EXTRACT, DATA_TRUNC, TIMEOFDAY, INTERVAL, serão vistas com mais detalhes no tutorial.

Um exemplo de função com data:

A cláusula AGE(timestamp, timestamp), retornará a subtração dos argumentos, produzindo um resultado "simbólico" que utiliza anos e meses.

```
SELECT AGE(timestamp '2001-04-10', timestamp '1957-06-13'),
AGE(timestamp '2001-04-10')  FROM nome_tabela;
```

O resultado será: AGE(timestamp '2001-04-10', timestamp '1957-06-13'),
43 years 9 mons 27 days
AGE(timestamp '2001-04-10'), este retornará a subtração
de CURRENT_DATE (data atual).

A cláusula CURRENT_DATE retornará a data atual.

ANEXO D - PRINCIPAIS COMANDOS SQL DO ORACLE

De acordo com FERNANDES (2002) e MORELLI (2002), os comandos de SQL no Oracle deve ser inserindo no SQLPlus ou SQL*Plus Worksheet, ferramenta instalada com o banco de dados, para executar todas as classes de comandos SQL, possuindo comandos padrão do SQL e os que são desenvolvidos para BD Oracle. O padrão suportado é o ANSI/ISO SQL-92 (ANSI – ***American National Standards Intitute***, ISO – ***International Organization of Standards***). (ORACLE)

O formato de dados suportados no Oracle serão apresentados na Tabela D.1:

Tabela D.1 - Mostra a lista de tipos de dados suportados no Oracle.

Nome	Tamanho	Faixa / Precisão	Descrição
Char (size)[BYTE | CHAR]	2000 bytes	tamanho mínimo de 1 byte	número de caracteres em uma cadeia
Varchar2	4000 bytes	1 a 4,096	cadeia de caracter com tamanho variável
Clob	4 gigabytes	precisão de 6 dígitos decimais	dados grandes e não estruturados,
Long	2 gigabytes	variável até 2 gigabytes, ou 231-1 bytes	comprimento variável de dados
Blob	4 gigabytes	-	grande objeto binário
Raw	2000 bytes	-	números binários
Long Raw	2 gigabytes	-	dados binários variáveis
Number(p,s)	-	precisão p de (1 a 38) e s (-84 a 127) 7456123.89	número de precisão
Date	-	1 janeiro, 4712 AC até o 31 de Dezembro, 4712 DC	somente datas
Rowid	16 bytes	-	cadeia hexadecimal
Timestamp (precisão em segundos)	-	-	tanto data quanto hora

Para iniciar este aplicativo devemos ter o *login* e o *password*, quando não é definidos na instalação, são criados por *default* (*login*: scott, *password*: tiger).

Para se mudar a senha deve-se digitar:

```
SQL> ALTER USER nome_usuario IDENTIFIED BY nova_senha;
```

Para se listar todos os usuários;

```
SQL>  SELECT  * FROM ALL_USERS;
```

O comando CREATE DATABASE, cria um banco de dados.

```
SQL> CREATE DATABASE database (DATABASE_ID database_id,
DATABASE_SIZE max_bytes, EXTENT_SIZE npages);
```

A criação da base de dados pode ter os parâmetros de configuração, DATABASE_ID número opcional para identificação do banco de dados, DATABASE_SIZE tamanho do banco de dados, EXTENT_SIZE um montante incremental de páginas em um arquivo de banco de dados. Quando um banco de dados fica sem páginas no arquivo atual, ele estende o arquivo por este número de páginas.

O comando CREATE TABLE, cria as tabelas no banco de dados.

```
SQL> CREATE TABLE nome_tabela (
                    nome_campo1 VARCHAR2 (10) NOT NULL,
                    nome_campo2 INTERGER NULL,
                    nome_campo3 CHAR (5) NULL,
                    nome_campo4 NUMBER (5) NULL
                    );
```

Neste comando NOT NULL significa que o campo a ser inserido não pode ser nulo. O NULL para iniciar o campo como nulo.

Para inserir a chave primária nesta tabela usa-se o comando a seguir:

```
SQL> ALTER TABLE nome_tabela ADD (PRIMARY KEY (nome_campo));
```

Este comando pode inserir a chave estrangeira, que será visto a seguir;

```
SQL> ALTER TABLE nome_tabela ADD (FOREIGN KEY (nome_campo)
REFERENCES nome_tabela_referente);
```

REFERENCES significa que a tabela (nome_tabela) irá ter duas chaves primárias, a sua própria chave e uma outra, da tabela (nome_tabela_referente).

O comando ALTER TABLE, pode alterar e inserir uma coluna em uma tabela, que serão ilustrados a seguir:

```
SQL> ALTER TABLE nome_tabela ADD (nome_campo1 CHAR(10) NULL );
```

Ao inserir um campo novo em uma tabela, pode-se escolher os atributos (nome_campo1 CHAR(10) NULL).

Também se pode fazer a modificação de um campo com MODIFY ilustrado a seguir:

```
SQL> ALTER TABLE nome_tabela MODIFY (nome_campo VARCHAR (10)
NOT NULL);
```

Pode-se alterar a tabela mudando o nome da coluna, este será mostrado a seguir:

```
SQL> ALTER TABLE nome_tabela RENAME COLUMN nome_antigo
TO nome_novo;
```

Também se pode excluir um campo da tabela usando o DROP:

```
SQL> ALTER TABLE nome_tabela DROP (nome_campo);
```

O comando DROP elimina uma tabela, este será mostrado a seguir:

```
SQL> DROP TABLE nome_tabela;
```

O comando para adicionar linhas em uma tabela INSER INTO será ilustrado a seguir;

```
SQL> INSERT INTO nome_tabela (nome_campo1, nome_campo2, ...)
VALUES (valor_campo1, valor_campo2, ...);
```

Se o nome do campo não estiver definido no INSERT, os valores serão inserindo de forma que foram criados no CREATE TABLE. Os valores que não forem numéricos deverão ser escritos entre aspas (‘), exemplo: TO DATE (‘21-09-1978’, ‘dd-mm-yyyy’), ‘nome_campo’.

O comando para alterar o valor de uma tabela é UPDATE que será mostrado a seguir:

```
SQL> UPDATE nome_tabela SET nome_campo = ‘novo_campo’
        WHERE nome_campo = ‘campo_existente’;
```

Na instrução anterior será alterado o campo para “novo_campo”, onde a cláusula WHERE define a condição de onde o campo será alterado, se WHERE for omitido alterará todos os campos da tabela.

O comando DELETE remove linhas de uma coluna, que será ilustrado a seguir:

```
SQL> DELETE FROM nome_tabela WHERE nome_campo = ‘valor _campo’;
```

As transações de DML não serão atualizadas no BD até receber um comando COMMIT OU ROLLBACK. O COMMIT valida as transações pendentes e o ROLLBACK desfaz as transações pendentes, retornando os valores antes de se executar os comandos DML. Os comandos DDL não precisam ser validados.

O comando SELECT retorna o que foi inserido no bando de dados:

```
SQL> SELECT * FROM nome_tabela;
```

O asterisco (*) significa que será mostrado todo o campo da tabela nome_tabela. Podem-se escolher os campos para visualiza:

SQL> SELECT nome_campo1, nome_campo2. ... FROM nome_tabela;

Poderão inserir funções aritméticas nas consultas (+.-.*, /), com prioridade (*, /, +, -). Se tiver operadores iguais, a prioridade será da esquerda para a direita.

SQL> SELECT nome_campo1 + 2, nome_campo2 / 2, ... FROM nome_tabela;

O comando DISTINCT elimina valores duplicados em um ou todos os campos:

SQL> SELECT DISTINCT nome_campo FROM nome_tabela;

O comando ORDEY BY ordena os campos em ordem alfabética ou numérica:

SQL> SELECT nome_campo1, nome_campo2 FROM nome_tabela ORDEY BY nome_campo DESC;

O comando DESC ordena do maior para o menor, e ASC do menor para o maior.

A cláusula WHERE corresponde ao operador da álgebra relacional de restrição. Este deverá ser escrito logo após FROM, para selecionar as linhas que desejamos, podendo comparar valores de colunas, literais, expressões aritméticas ou funções.

SQL> SELECT nome_campo1, nome_campo2 FROM nome_tabela WHERE nome_campo = ‘nome’;

OBS.: *Strings* e datas deverão estar entre apóstrofos (‘’)

Os operadores podem ser lógicos (=,>,<,>=,<=) e operadores SQL (BETWEEN AND, IN, LIKE, IS NULL)

Operador BETWEEN testa os valores de um intervalo inferior e superior ao limite previamente especificado na cláusula, sendo o inferior antes do superior.

```
SQL> SELECT nome_campo1, nome_campo2 FROM nome_tabela WHERE
nome_campo  BETWEEN valor01 AND valor02;
```

Operador IN compara os valores em uma lista, string e datas deverão estar entre apóstrofos (‘’).

```
SQL> SELECT nome_campo1, nome_campo2 FROM nome_tabela
WHERE nome_campo IN (valor01, valor02, valor03,...);
```

Operador LIKE é usado quando o valor exato de uma *string* não é conhecido. Permite selecionar linhas cujas colunas correspondem em parte com o argumento a ser pesquisado, podendo usar coringas para construir uma string para pesquisa, os caracteres são “%” e “_” são usados para representar qualquer seqüência de zeros ou mais caracteres, e para qualquer caracter único, respectivamente.

```
SQL> SELECT nome_campo1, nome_campo2 FROM nome_tabela
WHERE nome_campo LIKE ´nome_letra%´;
```

O comando IS NULL testa valores que são nulos.

```
SQL> SELECT nome_campo1, nome_campo2 FROM nome_tabela
WHERE nome_campo IS NULL;
```

Podem-se ter operadores de negação para os comandos como NOT BETWEEN AND, NOT IN, NOT LIKE, IS NOT NULL visto a seguir.

```
SQL> SELECT nome_campo1, nome_campo2 FROM nome_tabela
WHERE nome_campo NOT BETWEEN valor01 AND valor02;
```

No comando SELECT pode-se inserir as cláusulas AND (todas as condições satisfeitas) e OR (pelo menos uma das condições satisfeitas), para filtrar ainda mais, de forma clara e não tendo aqueles campos que são desnecessários, visto a seguir.

> SQL> SELECT nome_campo1, nome_campo2 FROM nome_tabela WHERE nome_campo BETWEEN valor01 AND valor02 and nome_campo = ‘outro_campo’ OR nome_campo = ‘outro valor’;

Neste comando terá a procedência do comando AND sobre OR, isto significa que a cláusula AND será executada antes do OR. Nesta expressão podemos verificar que há procedência de operadores. Se houver operadores de mesma procedência, o comando será executado da esquerda para a direita. A ordem dos operadores SQL**:** =, <>,<, >, <=, >=, BETWEEN, IN, LIKE, IS NULL, depois o NOT, AND e por último o OR. Caso tenha dúvida utilizar os parênteses “()” para separar as cláusulas.

Funções são usadas para a manipulação de itens de dados, podendo aceitar um ou mais argumentos, retornando um valor do BD. O argumento pode ser uma constante, variável ou coluna.

As funções são utilizadas para executar cálculos em dados, modificar itens de dados individualmente, manipular saída de um grupo de linhas, alterar formato de datas para mostrá-los, converter tipos de dados das colunas. Os tipos de funções são: *caracter*, *number, conversion, date, group*.

As funções de linha podem ser usadas em qualquer lugar, que permitam variáveis de usuários, colunas e expressões, funções atuam em cada linha retornada pela *query*, retornando um resultado por linha.

Funções de caractere aceitam caractere como entrada e retornam caractere ou números, tais como funções de LOWER, INSTR UPPER, LTRIM, INITCAP, LCAP, LENGTH, RPAD, TRANSLATE, SUBSTR, REPLACE, serão detalhados no tutorial.

Um exemplo de função de caractere:

O comando LOWER transforma caracteres alfabéticos que estão em maiúsculos para minúsculos, visto a seguir.

```
SQL> SELECT LOWER(nome_campo) FROM nome_tabela;
```

Funções matemáticas aceitam números como entrada e retornam valores numéricos, tais como funções de ROUND, TRUNC, CEIL, FLOOR, POWER, SQRT, SIGN, ABS, MOD, POWER, serão detalhados no tutorial.

Um exemplo de função matemática:

O comando ROUND arredonda a coluna expressão ou valor para “n” casas decimais, se “n” for omitido as casas decimais serão eliminadas, se o “n” for negativo o numero será arredondado para zero, esquerda da vírgula.

```
SQL> SELECT ROUND(‘12.123’, 1), ROUND(‘12.123’.’-1’)
FROM nome_tabela;
```

As funções de manipulação de data, no Oracle as datas serão armazenadas em formato numérico interno, que pode variar de 1-jan-4712 AC até 31-dez-4712 DC. O formato pode variar em século, ano mês, dias horas, minutos e segundos. As funções de datas podem ser: SYSDATE, MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND, TRUNC(date, ‘MONTH’), TO_CHAR, serão detalhados no tutorial.

Um exemplo de manipulação de data:

O comando SYSDATE é um pseudocoluna que retorna a data e hora corrente no sistema. Normalmente e comum selecionar SYSDATE a partir de uma tabela vazia chamada DUAL.

```
SQL> SELECT SYSDATE FROM DUAL;
```

O resultado será: SYDATE

13/09/05

Também podemos realizar cálculos de datas com operadores aritméticos como:

DATE + NUMBER = adiciona um numero de dias à data, retornando uma data.

DATE – NUMBER = Subtrair um número de dias à data, retorna uma data.

DATE – DATE = subtrair uma data de outra, retornando o numero de dias.

DATE + NUMBER/24 = adiciona um numero de horas na data, e retorna uma data.

SQL> SELECT SYSDATE FROM DUAL;

O resultado será: SYDATE

10/09/05

Lista dos formatos de datas e horas ilustrado na Tabela D.2:

Tabela D.2 - Mostra a lista de formatos de datas e horas no Oracle.

SCC ou CC	Século; 'S' coloca o prefixo '-' para BC
YYYY ou SYYYY	Ano; 'S' coloca o prefixo '-' para BC
YYY ou YY ou Y	Últimos 3,2,1 dígitos do ano
Y, YYY	Ano com Virgula na posição indicada
SYEAR ou YEAR	Ano; 'S' coloca o prefixo '-' para BC
BC, AC ou B.C. ou A.D.	Indicador BC / AD
Q	Trimestre
MM	Mês
MONTH, MON	Nome do mês, com 9 caracteres, e 3 caracteres
WW ou W	Semana do ano ou mês
DDD ou DD ou D	Dia do ano, mês ou semana
DAY, DY	Nome do dia, com 9 caracteres, e 3 caracteres
J	Data Juliana, numero de dias desde 31-de-4713 AC
AM, PM ou A.M.	Indicador AM/PM
HH ou HH12	Hora do dia (1-12)
HH24	Hora do dia (1-24)
MI	Minutos
SS	Segundos
SSSSS	Segundos depois da meia-noite (0-86399)
/.	Pontuação reproduzida no resultado
"—"	String reproduzida no resultado
Fm	Prefixo de MONTH ou DAY para Eliminar espaços
TH	Sufixo para numero ordinal
SP	Sufixo para *spelled-out* (soletrar) o número (ex.: DDSP)
SPTH	Sufixo para *spelled-out* (soletrar) o número (ex.: DDSPTH)

O operador UNION retorna todas as linhas distintas recuperadas de uma QUERY.

```
SQL> SELECT nome_campo, nome_campo1 FROM nome_tabela
WHERE nome_campo = ‘nome’ UNION SELECT
nome_campo, nome_campo1 FROM nome_tabela
WHERE nome_campo = ‘nome’;
```

O operador INTERSECT retorna somente as linhas recuperadas em ambas das QUERY.

```
SQL> SELECT nome_campo, nome_campo1 FROM nome_tabela
WHERE nome_campo = ‘nome’ INTERSECT
SELECT nome_campo, nome_campo1
FROM nome_tabela WHERE nome_campo = ‘nome’;
```

O operador MINUS retorna somente o resultado que está presente na primeira QUERY e que não está no segundo resultado da QUERY.

```
SQL> SELECT nome_campo, nome_campo1 FROM nome_tabela
WHERE nome_campo = ‘nome’ MINUS
SELECT nome_campo, nome_campo1
FROM nome_tabela WHERE nome_campo = ‘nome’;
```

As funções de grupo operam em conjunto de linhas, o resultado e baseado em um grupo de linhas, ao invés de um resultado por linha. Os alguns dos operadores em grupo são: AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE, que serão detalhados no tutorial. Estes operadores poderão vir seguidos das seguintes cláusulas:

DISTINCT faz com que a função considere somente valores distintos.

ALL é o valor default, esta inclui os valores duplicados.

Os tipos de dados para estas expressões podem ser CHAR, NUMBER ou DATE.

OBS: Todas as funções ignoram valores NULL, exceto COUNT(*).

Um exemplo de operador de grupo:

O comando AVG resulta na média de “n” elementos, ignorando null.

```
SQL> SELECT AVG(nome_campo) FROM nome_tabela;
```

A cláusula GROUP BY pode ser usado para dividir um conjunto de linhas em grupos menores, enquanto que as funções de grupo são usadas para retornar informações sobre cada grupo.

```
SQL> SELECT nome_campo, AVG(nome_campo1)
FROM nome_tabela GROUP BY nome_campo;
```

Este comando ira fazer a média do nome_campo1, separando pelo nome_campo.

O comando GROUP BY pode selecionar as linhas, porém há linhas que não serão necessárias, então pode inserir a cláusula WHERE para não selecionar esta linha.

```
SQL> SELECT nome_campo, AVG(nome_campo1)
FROM nome_tabela WHERE nome_campo <> ‘nome’
GROUP BY nome_campo;
```

Pode-se usar a cláusulas GROUP BY para fornecer resultados de grupos dentro de grupos.

```
SQL> SELECT nome_campo, nome_campo1, AVG(nome_campo2)
FROM nome_tabela GROUP BY nome_campo, nome_campo1;
```

Este comando irá separar por nome_campo e depois por nome_campo1, fazendo a média do nome_campo.

A cláusula HAVING específica quais grupos devem ser mostradas na consulta, as cláusulas WHERE restringe as linhas que serão selecionadas para a formação dos grupos e o HAVING restringe os grupos já formados.

```
SQL> SELECT nome_campo, AVG(nome_campo1)
FROM nome_tabela GROUP BY nome_campo HAVING
CONT(*)  > ‘numero’;
```

O comando de SUBQUERY é usado para fazer uma consulta SELECT aninhada dentro de outro comando SELECT.

```
SQL> SELECT nome_campo, nome_campo1 FROM nome_tabela
WHERE nome_campo1 = (SELECT MAX(nome_campo1)
FROM nome_tabela);
```

O operador IN inserido no SELECT aninhado inclui mais que uma linha na consulta, podendo ser comparado mais de um valor.

```
SQL> SELECT nome_campo, nome_campo1 FROM nome_tabela WHERE
(nome_campo, nome_campo2) IN (SELECT MAX(nome_campo1),
nome_campo2 FROM nome_tabela GRUOP BY nome_campo2);
```

O operador ALL e ANY utilizado em SUBQUERY, que retornam mais de uma linha na consulta.

ALL retorna um valor comparado com todas os valores retornados pela SUBQUERY.

ANY retorna um valor comparado para cada valor retornado pela SUBQUERY.

```
SQL> SELECT nome_campo, nome_campo1, nome_campo2 FROM
 nome_tabela WHERE nome_campo1 >
ANY (SELECT DISTINCT nome_campo1
FROM nome_tabela WHERE nome_campo2 = ‘numero’);
```

O comando JOINS é usado quando utilizam dados de mais de uma tabela, e as linhas de uma tabela podem ser juntadas com linhas de outra tabela, o JOINS é especificado na cláusula WHERE, a forma mais simples de representar e o EQUIJOIN, quando os valores da tabela que

são iguais ao desejados da outra tabela, isto se a igualdade de chaves corresponde com o modelo-relacional.

Se o for omitido, ou se estiver errada a cláusula JOIN, o resultado poderá ter várias linhas que serão inúteis, este resultado será o produto cartesiano da álgebra relacional.

```
SQL> SELECT nome_campo, nome_campo1, nome_campo2
FROM nome_tabela, nome_tabela1
WHERE nome_tabela.codigo = nome_tabela1.codigo;
```

No comando anterior o código das duas tabelas deverá ter a igualdade de chaves.

O comando ALIASES é utilizado para simplificar os comandos SQL, os nomes das tabelas poderão ser trocados em FROM, então este nome poderá ser usado nas demais cláusulas da consulta. Estes nomes podem ser grandes podendo ter 30 caracteres, mais para facilitar coloca-se apenas uma letra.

```
SQL> SELECT A.nome_campo, A.nome_campo1, B.nome_campo2
FROM nome_tabela A, nome_tabela1 B WHERE A.codigo = B.codigo;
```

O comando NON-EQUI_JOIN em alguma das vezes não se tem colunas comuns nas tabelas envolvidas em um relacionamento, então se utiliza o operador BETWEEN que não consiste em um relacionamento de igualdade.

```
SQL> SELECT A.nome_campo, A.nome_campo1, B.nome_campo2
  FROM nome_tabela A, nome_tabela1 B
  WHERE A.nome_campo1 BETWEEN B.nome_campo3 AND B.nome_campo4;
```

Pode-se fazer um JOIN na mesma tabela, isto permite a junção de linhas da tabela em uma mesma tabela, visto a seguir.

```
SQL> SELECT A.nome_campo * novo_nome*, A.nome_campo1 *novo_nome1*,
B.nome_campo2 *novo_nome2* B.nome_campo3 *novo_nome3*
FROM nome_tabela A, nome_tabela1 B WHERE
A.nome_campo = B.nome_campo AND A.nome_campo > B. nome_campo;
```

Os operadores de conjunto são úteis em QUERY que referenciam diversas tabelas, combinando resultados de varias tabelas em um único resultado, por isso e conhecido com joins verticais. Tem que obedecer as seguintes regras nestes operadores de conjunto:

- O comando SELECT deverá selecionar os mesmos conjuntos e será executado de cima para baixo;
- As colunas devem ter o mesmo DATATYPE (o mesmo tipo);
- As linhas duplicadas serão eliminadas e não poderá utilizar a cláusula DISTINCT;
- Os nomes da coluna da primeira QUERY que serão mostrados no resultado;
- A cláusula ORDEY BY poderá aparecer no final do comando;
- Poderão ser usados os operadores de conjunto em SUBQUERIES;
- Os operadores poderão ser usados em vários lugares desde que estejam entre parênteses.