

# *Oracle* *Básico*



**Marcelino Saraiva Mota**

<b>INTRODUÇÃO AO BANCO DE DADOS</b>	<b>7</b>
<b>Conceitos Básicos de Banco de Dados</b>	<b>8</b>
<b>O Que é um Sistema de Banco de Dados?</b>	<b>8</b>
Dados	8
Hardware	8
Software	8
Usuários	8
<b>Modelos de Banco de Dados</b>	<b>10</b>
<b>Hierárquico</b>	<b>10</b>
<b>Rede</b>	<b>10</b>
<b>Relacional</b>	<b>11</b>
Arquitetura Banco De Dados Oracle	12
<b>Introdução ao Oracle e à Linguagem SQL</b>	<b>14</b>
<b>Sistema de Gerenciamento de Banco de Dados</b>	<b>15</b>
<b>O Que é um Banco de Dados?</b>	<b>15</b>
<b>O Sistema Gerenciador de Banco de Dados</b>	<b>15</b>
<b>O Acesso Relacional</b>	<b>15</b>
<b>Conceitos Básicos</b>	<b>15</b>
▪ COMANDO	15
▪ BLOCO	15
▪ TABELA	15
▪ PESQUISA	16
▪ RELATÓRIO	16
<b>Operadores Relacionais</b>	<b>16</b>
<b>Propriedades de um Banco Relacional</b>	<b>19</b>
<b>As Propriedades de uma Única Tabela</b>	<b>19</b>
<b>Arquitetura dos Produtos Oracle</b>	<b>20</b>
<b>A Interface SQL*PLUS</b>	<b>20</b>
Chamando o SQL*Plus	20
Conectando ao Banco	21
<b>Comandos do Editor do SQL*Plus</b>	<b>21</b>
<b>Comandos Genéricos do SQL*Plus</b>	<b>22</b>
<b>Introdução à Linguagem SQL</b>	<b>25</b>
<b>Visão Geral Sobre o SQL</b>	<b>25</b>
<b>Características do SQL</b>	<b>25</b>
▪ DEFINIÇÃO DE DADOS	25
▪ MANIPULAÇÃO DE DADOS	25
<b>Escrevendo Comandos SQL</b>	<b>25</b>
<b>Bloco de Pesquisa Básico</b>	<b>27</b>
<b>Expressões Aritméticas</b>	<b>27</b>
<b>Apelidos De Colunas</b>	<b>28</b>

<b>Operador De Concatenação</b>	<b>29</b>
<b>Literais</b>	<b>30</b>
<b>Trabalhando Com Valores Nulos</b>	<b>31</b>
<b>Outros Itens Do Select (Cláusulas)</b>	<b>32</b>
Prevenindo a Seleção de Linhas Duplicadas (DISTINCT)	32
Ordenando a Saída do SELECT (ORDER BY)	33
Agrupando Dados no SELECT (GROUP BY)	34
Restringindo Os Grupos De Dados No Select (Having)	35
<b>Realizando restrições no SELECT (WHERE)</b>	<b>36</b>
Operadores Lógicos	36
= Igual a	36
> Maior que	37
>= Maior ou igual a	37
< Menor que	37
<= Menor ou igual a	38
<b>Operadores SQL</b>	<b>39</b>
BETWEEN .. AND	39
IN (list)	40
LIKE	41
IS NULL	42
Seleciona os dados que tenham valor nulo.	42
Expressões De Negação	42
Substituição De Variáveis	43
&	43
&&	43
DEFINE	44
<b>Pesquisando Dados Com Múltiplas Condições</b>	<b>45</b>
Precedência dos Operadores	46
SUMARIO DO SELECT	46
<b>Funções</b>	<b>47</b>
<b>Funções Caracter De Linha Simples Com Retorno De Caracter</b>	<b>47</b>
LOWER	47
UPPER	47
INITCAP	48
LPAD	48
RPAD	48
SUBSTR (col   valor, pos, n)	49
LTRIM (col   valor, 'char')	49
RTRIM (col   valor, 'char')	49
SOUNDEX(col   valor)	50
TRANSLATE (col   valor, from, to)	50
REPLACE(col   valor, string, string_substituta)	50
<b>Funções Caracter De Linha Simples Com Retorno De Número</b>	<b>51</b>
LENGTH (col   valor)	51
INSTR (col   valor, 'string', pos, n)	51
<b>Funções Numéricas De Linha Simples Com Retorno De Número</b>	<b>52</b>
ROUND (col   valor, n)	52
TRUNC(col   valor, n)	52
CEIL (col   valor)	52
FLOOR (col   valor)	53
POWER (col   valor,n)	53

ABS (col   valor)	53
MOD (valor1, valor2)	53
SQRT (col   valor)	53
SIGN(col   valor)	53
<b>Funções De Data</b>	<b>55</b>
MONTHS_BETWEEN(data1, data2)	55
ROUND(data1, char)	55
TRUNC(data1,char)	55
ADD_MONTHS(data,n)	56
NEXT_DAY(data1,char)	56
LAST_DAY(data1)	56
<b>Funções De Conversão</b>	<b>57</b>
TO_CHAR (número   data, 'mascara edição')	57
TO_NUMBER (char)	57
TO_DATE (número   data, 'mascara edição')	57
FORMATOS DE DATA	58
<b>Outras Funções</b>	<b>59</b>
GREATEST(col   valor1, col   valor2, ...)	59
VSIZE(col   valor)	59
DECODE(col   expressão, search1, result1, default)	59
NVL(col   valor, valor1)	60
<b>Funções De Grupo</b>	<b>61</b>
AVG([distinct   all] n)	61
COUNT([distinct   all expr *])	61
MAX([distinct   all expr])	61
MIN([distinct   all expr])	61
STDDEV([distinct   all n])	62
SUM([distinct   all n])	62
<b>Pseudo-Colunas</b>	<b>63</b>
<b>SEQUENCES</b>	<b>63</b>
CURRVAL	63
NEXTVAL	63
<b>SYSDATE</b>	<b>63</b>
<b>LEVEL</b>	<b>63</b>
<b>ROWID</b>	<b>64</b>
<b>ROWNUM</b>	<b>65</b>
<b>DUAL</b>	<b>66</b>
<b>USER</b>	<b>66</b>
<b>UID</b>	<b>66</b>
<b>Junção</b>	<b>67</b>
<b>Equi-Join</b>	<b>67</b>
Uso De Aliases Para Nomes De Tabelas	69
<b>Non-Equi-Join</b>	<b>70</b>
<b>Outro métodos de junção</b>	<b>71</b>
<b>Outer Join</b>	<b>71</b>

Oracle Básico	5
Join Reflexivo	72
<i>Operadores de Conjunto</i>	73
Union	73
Intersect	74
Minus	74
OBSERVAÇÕES	74
<i>Subqueries</i>	75
Subquery Com Retorno De Uma Única Linha	75
Subquery Com Retorno De Mais De Uma Linha	75
ANY	76
ALL	76
HAVING	77
EXIST	77
<i>Comandos D.D.L.</i>	78
Criação De Tabelas	81
TIPOS DE DADOS	82
CONSTRAINT PARAMETROS	82
<i>DATA DICTIONARY</i>	86
<i>Comandos D.M.L.</i>	94
Inserindo Dados Na Tabela	94
Alterando Dados Na Tabela	95
Excluindo Linhas Da Tabela	96
<i>Controle de transações</i>	98
Controlando As Transações Com Comandos Sql	98
COMMIT	98
SAVEPOINT	98
ROLLBACK	99
<i>DESENVOLVIMENTO DE APLICAÇÕES com PROCEDURAL OPTIONS</i>	101
<i>Conceitos Básicos</i>	102
<i>Blocos em PL/SQL</i>	108
<i>Manipuladores de Exceções</i>	109
<i>Comandos do PL/SQL</i>	112
<i>Desenvolvendo STORED PROCEDURES e Funções</i>	119
Criação de Procedures com tipo IN de argumentos	121
Eliminando Parâmetros Desnecessários de Entrada	122
Criação de Procedures com tipo OUT de argumentos	123
Criação de Procedures com tipo IN OUT de argumentos	124
Exceção Definida Pelo Usuário	128
Exceção definida pelo ORACLE	129
<i>Desenvolvendo e Utilizando PACKAGES</i>	136
<i>Desenvolvendo DATABASE TRIGGERS</i>	141

## Linguagem Consultoria e Treinamento

---

Oracle Básico	6
<i>Exercícios</i>	<i>149</i>
<i>RESPOSTAS DOS EXERCÍCIOS</i>	<i>154</i>

---

# *Capítulo 1*

## **INTRODUÇÃO AO BANCO DE DADOS**

# Conceitos Básicos de Banco de Dados

## O Que é um Sistema de Banco de Dados?

Há alguns anos atrás, em nossas primeiras publicações, havíamos adotado o seguinte conceito para definir um sistema de banco de dados:

*Um banco de dados é um reservatório contendo todos os itens de dados de um sistema informático, mantendo-se em disponibilidade para quem quiser acessar ou buscar informações.*

Hoje esse conceito foi ampliado para:

*Um sistema de manutenção de informações por computador consideradas como significativas ao indivíduo ou à organização servida pelo sistema, cujo objetivo global é manter as informações atualizadas e torná-las disponíveis quando solicitadas para o processo de tomada de decisão.*

Com a ampliação do conceito, notamos que o computador deixou de ser um depositário de informações, muitas vezes mal aproveitadas, para se transformar numa ferramenta de apoio à decisão, fornecendo informações cada vez mais precisas e mais rápidas.

Um sistema de banco de dados engloba quatro componentes básicos: dados, hardware, software (DBMS) e usuários.

### Dados

São as informações que ficam armazenadas no banco de dados e que são compartilhadas (ao mesmo tempo ou não) por todos os usuários do banco.

Quando dizemos que um banco de dados é **INTEGRADO**, estamos dizendo que existe uma unificação entre as diversas informações com um mínimo de redundância entre as mesmas.

E, **COMPARTILHADO**, quando as diferentes informações isoladas ou em conjunto podem ser manipuladas ao mesmo tempo por mais de um usuário.

### Hardware

É a parte física do banco de dados; são os discos utilizados para armazenamento do banco de dados, juntamente com os dispositivos de entrada e saída de informações (monitores, impressoras etc.).

### Software

É a parte do controle sobre as informações contidas no banco de dados, isto é, feito por um sistema ao qual damos o nome de **SISTEMA GERENCIADOR DE BANCO DE DADOS** ou DBMS (Data Base Management System).

O DBMS monitora os dados sobre o seu controle, restringindo de modo que o acesso a eles seja feito por seu intermediário, isolando o usuário dos detalhes a nível de hardware. Isto significa que o usuário não precisa saber onde está armazenada a informação que lhe pertence, e tampouco o tipo de hardware utilizado nesta função.

### Usuários

Os usuários, por sua vez, estão agrupados em classes: desenvolvedor de aplicações, usuário final e administrador de banco de dados.

#### ➤ DESENVOLVEDOR DE APLICAÇÕES



Esta classe compreende mais uma subdivisão, que é: **DESENVOLVEDOR** e **CASE**.

O **DESENVOLVEDOR** é o usuário responsável pela criação dos programas que irão ser executados pela emissão de solicitações apropriadas ao DBMS, tais como:

- Recuperação de informações,
- Criação de novas informações
- Anulação ou alteração das informações existentes.

É quem tem contato direto com o usuário final e faz a ponte entre ele e o Case.

O **CASE** é o usuário responsável pela modelagem dos dados e implementação do banco de dados nos níveis lógico e físico junto com o DBA.

### ➤ **USUÁRIO FINAL**

É o usuário que faz uso das informações contidas no banco de dados para tomada de decisão ou simples consulta ou atualização.

### ➤ **ADMINISTRADOR DE BANCO DE DADOS (DBA)**

É o usuário que tem as seguintes responsabilidades:

- Instalar e atualizar o banco de dados e as ferramentas de aplicação;
  - Alojamento de sistemas de armazenamento e planejar os futuros armazenamentos de requerimentos para o sistema de banco de dados;
  - Criação e armazenamento das estruturas primárias para que os desenvolvedores possam gerar aplicações;
  - Criação de objetos primários, uma vez que os usuários tenham construído uma nova aplicação;
  - Modificar a estrutura do banco de dados para adequá-lo às novas aplicações;
  - Registrar os usuários e manter o sistema de segurança;
  - Controlar e monitorar os acessos dos usuários ao banco de dados;
  - Fazer o backup das informações e a restauração;
  - Manter um controle sobre os backup's.
-

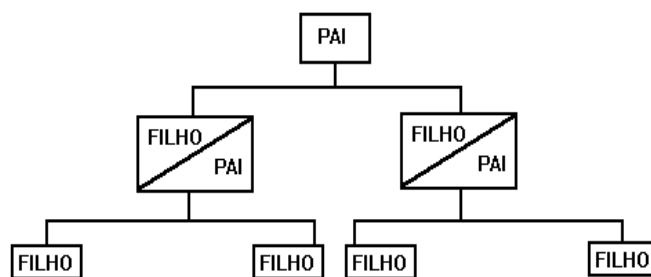
# Modelos de Banco de Dados

Um gerenciador de banco de dados pode basear-se em três diferentes formas de armazenamento para efetuar o controle de suas informações, essas formas são as seguintes: hierárquico, rede, relacional.

## Hierárquico

Em um modelo hierárquico, os dados que serão gravados obedecem a certos critérios pré-estabelecidos, que indicam qual o relacionamento do novo registro com os demais já existentes. Por ocasião da gravação são criados indicadores que possibilitam ao modelo determinar qual o próximo passo a seguir. Este relacionamento encadeia cada um dos dados do banco, através de listas hierárquicas num relacionamento de pai-filho.

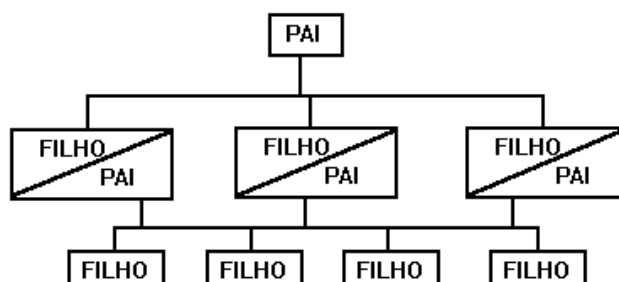
Todos os softwares, baseados em modelos hierárquicos, usam de artifícios tais como ponteiros e índices para não depender inteiramente da hierarquia. Caso contrário, o acesso a segmentos de nível muito baixo seria muito demorado, sendo impraticáveis também associações rápidas entre duas árvores.



## Rede

O modelo de rede é semelhante ao hierárquico, cada segmento possui um único pai, a diferença é que cada pai pode ter um ou mais filhos: um registro pode ter mais de um registro ascendente ligado a ele.

Dessa forma, a abordagem do modelo em rede parte diretamente para os níveis de organização de registros, inclusive de dados pertencentes à estrutura de acesso.

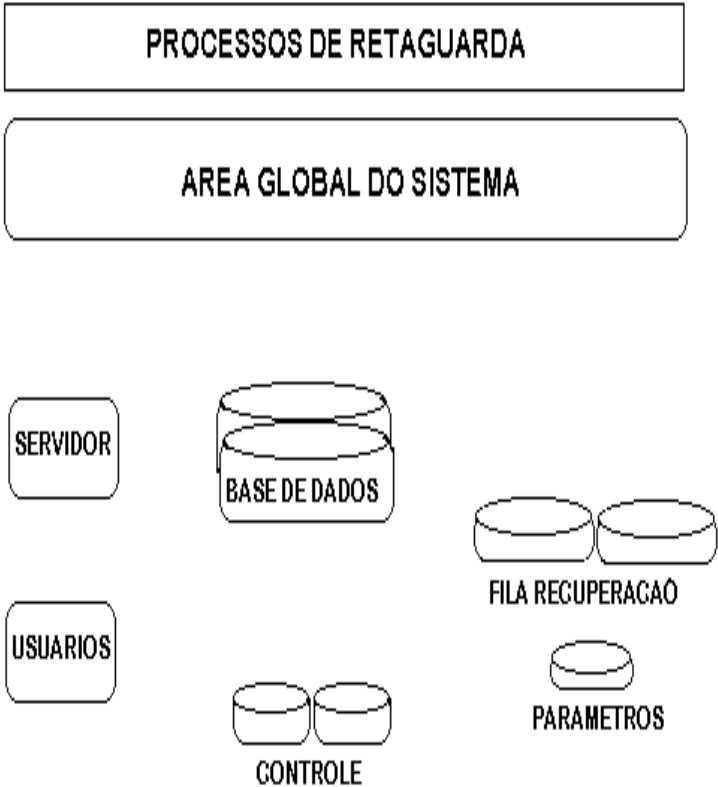


## Relacional

Os dados, neste modelo, são tratados como em uma matriz, onde cada campo de informação corresponde a uma coluna e cada registro, a uma linha. Dessa forma todos os registros sempre possuem todos os campos.

<b>NOME</b>	<b>ENDEREÇO</b>	<b>BAIRRO</b>	<b>CIDADE</b>
<b><i>Débora</i></b>	Rua A No 1	Aldeota	Fortaleza
<b><i>Diego</i></b>	Rua B No 2	Aldeota	Fortaleza
<b><i>David</i></b>	Rua C No 3	Aldeota	Fortaleza
<b><i>Isaac</i></b>	Rua D No 4	Centro	Fortaleza
<b><i>Samuel</i></b>	Rua E No 5	Centro	Fortaleza
<b><i>Erica</i></b>	Rua F No 6	Centro	Fortaleza
<b><i>Lucas</i></b>	Rua G No 7	City 2000	Fortaleza
<b><i>Pedro</i></b>	Rua H No 8	City 2000	Fortaleza
<b><i>Levy</i></b>	Rua I No 9	City 2000	Fortaleza
<b><i>Bruno</i></b>	Rua J No 10	Centro	Fortaleza

Arquitetura Banco De Dados Oracle



[illegible]

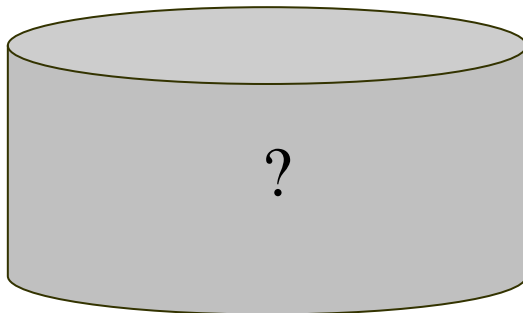
# *Capítulo 2*

## **Introdução ao Oracle e à Linguagem SQL**

# **Sistema de Gerenciamento de Banco de Dados**

## **O Que é um Banco de Dados?**

Um sistema de manutenção de informações por computador, consideradas como significativa ao indivíduo ou a organização servida pelo sistema, cujo objetivo global é manter as informações atualizadas e torná-las disponíveis quando solicitadas para o processo de tomada de decisão.



## **O Sistema Gerenciador de Banco de Dados**

Sistema Gerenciador de banco de dados ou DBMS (Data Base Management System) monitora os dados sobre o seu controle, de forma que todo acesso a eles seja feito por seu intermédio, isolando o usuário dos detalhes a nível de hardware. Isto significa que o usuário não precisa saber onde está armazenada a informação que lhe pertence, tampouco o tipo de hardware utilizado nesta função.

## **O Acesso Relacional**

Os princípios do modelo relacional foram impostos por Dr. E.F. Codd, em junho de 1970 no livro "A Relational Model of Data for Large Shared Data Banks". Neste livro, o Dr. Codd propõe o modelo "Relacional" para os sistemas de banco de dados.

O banco de dados relacional é compreendido pelos usuários como uma coleção de tabelas, que tem várias colunas (campos) e linhas.

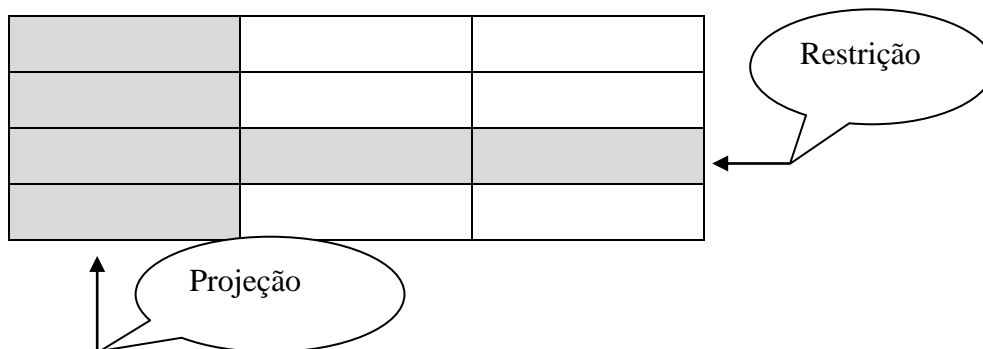
## **Conceitos Básicos**

- **COMANDO**  
Instruções disponíveis no SQL\*Plus.
- **BLOCO**  
Grupo de comandos SQL e PL/SQL de maneira procedural.
- **TABELA**  
Unidade básica de dados no RDBMS-ORACLE.

- **PESQUISA**  
Comandos SQL que recuperam informações de uma ou mais tabelas do RDBMS-ORACLE.
- **RELATÓRIO**  
Informações formatadas por comando SQL-Plus extraídas do RDBMS-ORACLE.

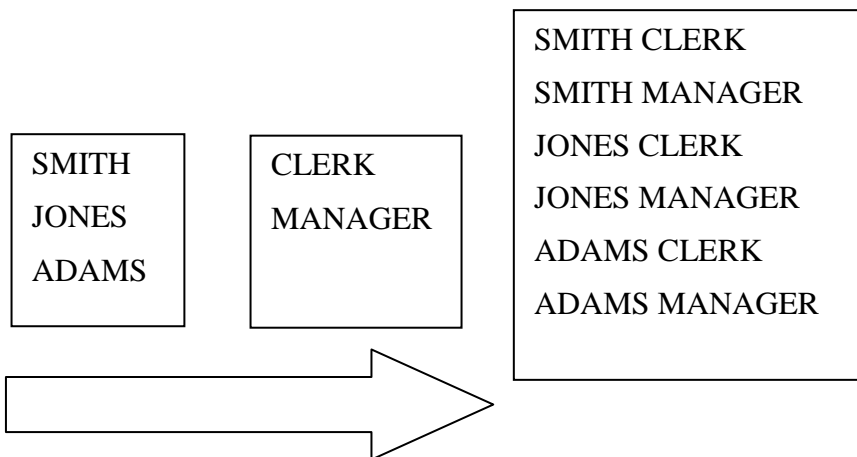
## Operadores Relacionais

OPERADOR	DESCRIÇÃO
Restriction	Restrição utilizada para filtrar os dados, é sempre a nível de linha. Chamado de subconjunto horizontal.
Projection	Projeta os dados da tabela, é sempre a nível de coluna. Chamado de subconjunto vertical.
Product	Resultado da concatenação dos subconjuntos (horizontal e vertical)
Join	O mesmo que o anterior. A diferença é que, neste caso, há uma condição especificada.
Union	Lista todas as linhas da união de ambas relações entre as tabelas.
Intersection	Lista todas as linhas da interseção entre duas tabelas
Difference	Lista a diferença entre duas tabelas

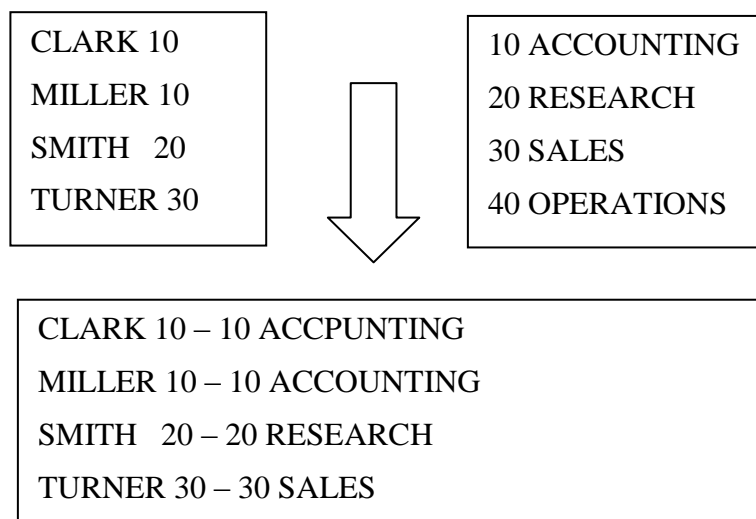




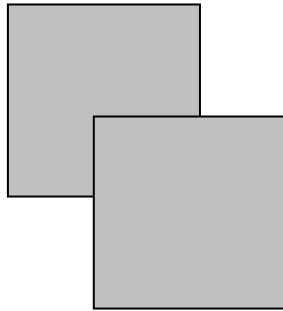
# PRODUTO



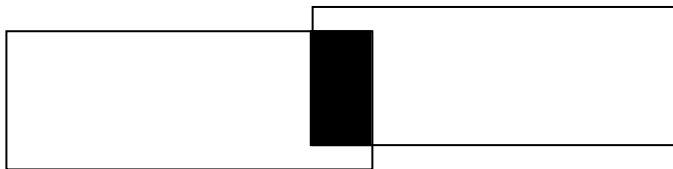
# JUNÇÃO



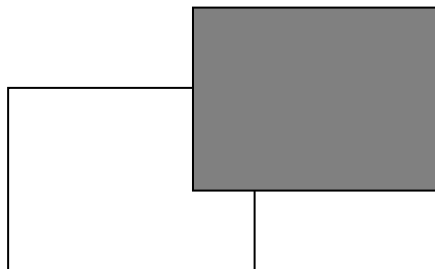
**UNIÃO**



**INTERSEÇÃO**



**DIFERENÇA**



## Propriedades de um Banco Relacional

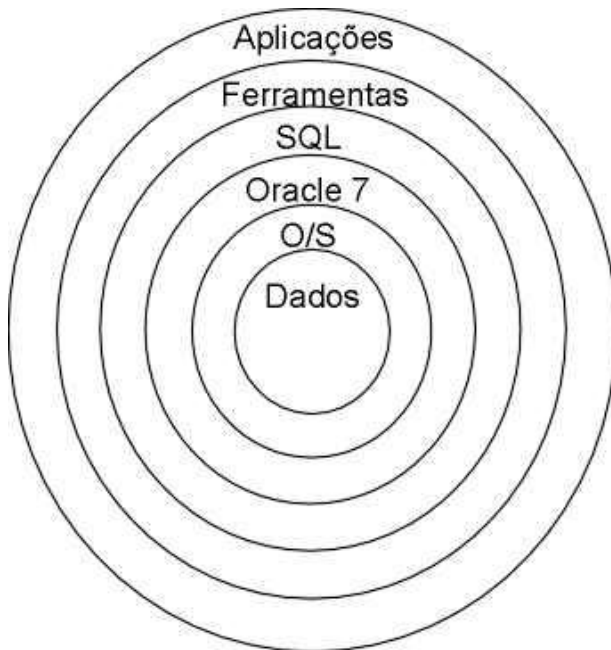
- É um conjunto de tabelas dos usuários;
- O formato de linhas e colunas da tabela é similar à visualização dos dados;
- Existência de um conjunto de operadores para particionar e combinar as relações entre as tabelas;
- Não tem ponteiros explícitos, as conexões são feitas puramente com os dados das tabelas;
- A linguagem Inglesa utilizada para pesquisa é não procedural;
- Os usuários não precisam especificar o caminho para acesso aos dados e nem saber como os dados estão armazenados;
- Os comandos de recuperação e atualização de dados são incluídos na linguagem SQL.



## As Propriedades de uma Única Tabela

- Não existem linhas duplicadas;
- Não existem nomes de colunas duplicadas;
- A ordem das linhas é insignificante;
- A ordem das colunas é insignificante;
- Os valores são atômicos (não se decompõem).

## Arquitetura dos Produtos Oracle



## A Interface SQL\*PLUS

O SQL\*Plus é uma ferramenta ORACLE que permite a interface interativa com o RDBMS-ORACLE, dirigida por comandos e usada para manipular a Base de Dados.

O SQL\*Plus permite utilizar comandos SQL, comandos SQL\*Plus juntamente com os comandos da linguagem procedural PL/SQL na montagem de procedures que necessitam de uma estrutura lógica de execução.

Com o SQL\*Plus podemos:

- Criar tabelas no RDBMS-ORACLE.
- Inserir informações nas tabelas.
- Recuperar informações formatadas das tabelas.
- Alterar informações nas tabelas.

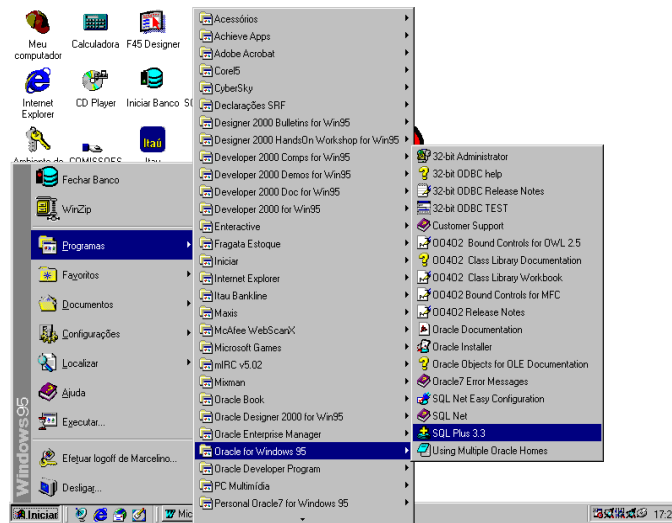
### Chamando o SQL\*Plus

Existem duas formas de se executar o SQL\*Plus:

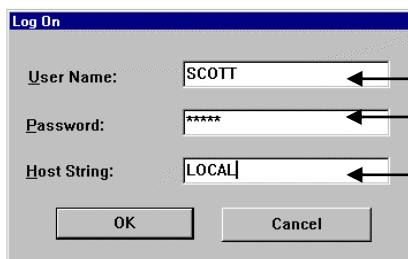
- O SQL\*Plus é chamado a partir do "prompt" do sistema operacional hospedeiro, para isto deve-se utilizar a seguinte sintaxe:

```
SQLPLUS [ [ - S [ ILENT]] [logon] [ start] ] | - ?  
logon: usuário [/ senha] [ @database - espec ] | / | / NOLOG  
start: arquivo contendo parâmetros e argumentos,  
.ext] [arg 1 arg 2 ....]
```

- SQL\*Plus é executado a partir do WINDOWS.



## Conectando ao Banco



Nome do Usuário

Senha de Acesso

Alias que define em que banco será conectado

## Comandos do Editor do SQL\*Plus

COMANDO	ABREVIATURA	DESCRIÇÃO
APPEND text	A text	Adiciona texto no final da linha corrente.
CHANGE	C/old/new	Troca texto old pelo texto new na linha corrente.
CHANGE	C/text/	Apaga o texto da linha corrente.
CLEAR BUFFER	CL BUFF	Apaga as linhas do BUFFER do SQL*Plus.
DEL		Apaga a linha corrente.
INPUT	I	Insere um número indefinido de linhas.
INPUT	I text	Insere linha fazendo consistência do texto.
LIST	L	Lista as linhas do BUFFER do SQL*Plus.
LIST n	L n	Lista a linha especificada em n
LIST m n	L m n	Lista de m até n.
RUN	R	Mostra e executa os comandos SQL do BUFFER.
/	/	Executa os comandos no BUFFER.

SAVE filename		Salva os comandos do BUFFER para um arquivo especificado.
GET filename		Coloca um arquivo de comandos SQL no BUFFER.
START filename		Executa o arquivo de comandos salvo previamente.
ED filename		Utiliza o editor do sistema operacional para trabalhar o arquivo de comandos SQL.
SPOOL filename		Grava os comandos subseqüentes em um arquivo para ser listado.
SPO[OL] OFF   OUT	SPO	OFF fecha o arquivo. OUT manda para a impressora.
DESC[RIBE] tablename	DESC	Mostra a estrutura da tabela
HELP		Chama o HELP do ORACLE.
HOST command		Executa um comando do sistema operacional.
CONN[ECT] userid/password	CONN	Conecta ao banco de dados.
EXIT		Sai do SQL*Plus.
PROMPT text		Mostra um texto durante a execução de um bloco de comandos.

## Comandos Genéricos do SQL\*Plus

- **@**  
Roda um arquivo de comando.
- **#**  
Termina uma seqüência de linhas de comentário iniciada por um comando DOCUMENT.
- **\$**  
Executa uma linha de comando do sistema operacional hospedeiro sem sair do SQL\*Plus.
- **/**  
Roda o comando no buffer do SQL.
- **APPEND**  
Anexa texto ao final da linha corrente no buffer corrente.
- **BREAK**  
Especifica que eventos causarão uma interrupção, e que ação o SQL deve executar em uma interrupção.
- **BTITLE**  
Faz com que o SQL mostre um título no fundo de cada página de um relatório.
- **CHANGE**  
Muda o conteúdo da linha corrente do buffer corrente.
- **CLEAR**  
Anula definições de interrupção, texto do buffer corrente, definições de coluna etc.
- **COLUMN**  
Especifica como uma coluna e um cabeçalho de coluna devem ser formatados em um relatório.
- **COMPUTE**  
Executa cálculos sobre grupos de linhas selecionadas.
- **CONNECT**

Serve para se conectar com a base de dados.

- **DEFINE**  
Define uma variável do usuário e atribui um valor char.
- **DEL**  
Deleta a linha corrente do buffer corrente.
- **DESCRIBE**  
Apresenta uma rápida descrição de uma tabela.
- **DISCONNECT**  
Submete o trabalho suspenso ao banco de dados e desconecta o usuário do RDBMS\_ORACLE, mas não termina o SQL\*Plus.
- **DOCUMENT**  
Inicia um bloco de documentação em um arquivo de comando.
- **EDIT**  
Chama o editor de texto padrão do sistema hospedeiro com o conteúdo do buffer corrente ou de um arquivo.
- **EXIT**  
Termina o SQL\*Plus e retorna o controle ao sistema operacional.
- **GET**  
Carrega o arquivo no buffer corrente.
- **HELP**  
Mostra informação sobre um comando do SQL ou do SQL\*Plus.
- **HOST**  
Executa um comando do sistema operacional hospedeiro sem sair do SQL\*Plus.
- **INPUT**  
Inclui novas linhas após a linha corrente do buffer corrente.
- **LIST**  
Lista linhas do buffer corrente.
- **NEWPAGE**  
Avança a saída em spool para o início da página seguinte.
- **PAUSE**  
Apresenta uma mensagem, depois espera que se tecele Enter.
- **QUIT**  
Termina o SQL\*Plus e retorna o controle ao sistema operacional, o mesmo que o EXIT.
- **REMARK**  
Inicia um comentário no programa.
- **ROLLBACK**  
Volta atrás (desfaz) mudanças feitas ao banco de dados default desde que foram submetidas pela última vez.
- **RUN**  
Apresenta e roda o comando no buffer do SQL.

- **SAVE**  
Salva o conteúdo do buffer corrente (um comando SQL ou um programa) no banco de dados ou em um arquivo do sistema operacional.
- **SET**  
Define uma variável do sistema com valor especificado.
- **SHOW**  
Apresenta o valor de uma variável do sistema ou de uma propriedade do SQL\*Plus, como o número da versão corrente.
- **SPOOL**  
Gerencia o spooling da saída na tela para um arquivo do sistema ou impressora do sistema.
- **SQLPLUS**  
Comando do sistema que inicia o SQL\*Plus.
- **START**  
Executa o conteúdo de um arquivo de comandos.
- **TIMING**  
Faz análise de performance em comandos SQL e programas SQL\*Plus.
- **TTITLE**  
Faz com que o SQL mostre um título no topo de cada página de saída.
- **UNDEFINE**  
Deleta a definição de uma variável do usuário.



# Introdução à Linguagem SQL

## Visão Geral Sobre o SQL

A linguagem SQL (Strutured Query Language) ou Linguagem Estruturada de Pesquisa, é composta por um grupo de facilidades para definição, manipulação e controle de dados em um banco de dados relacional.

A função da linguagem SQL é dar suporte à definição, manipulação e controle dos dados em um banco de dados relacional. Um banco de dados relacional é simplesmente um banco de dados que é percebido pelo usuário como um grupo de tabelas - onde uma tabela é um **grupo não ordenado de linhas** (relação é apenas o termo matemático para esse tipo de tabela).

Cada tabela pode ser considerada como um arquivo, com as linhas representando os registros e as colunas, os campos. O padrão SQL sempre usa os termos linha e coluna, nunca usando registro e campo.

## Características do SQL

Os comandos do SQL dividem-se em dois grupos:

- **DEFINIÇÃO DE DADOS**

São comandos que têm um fim de transação implícito (COMMIT), tais como: CREATE, ALTER.

- **MANIPULAÇÃO DE DADOS**

São comandos que necessitam de um fim de transação (COMMIT), tais como: INSERT, UPDATE, DELETE e SELECT

## Escrevendo Comandos SQL

Quando escrevemos um comando SQL, é importante lembrar algumas regras de construção de um comando válido e que seja fácil de ler e editar.

- Comandos SQL podem ser escritos em uma ou mais linhas.
- Cláusulas são usualmente escritas em linhas separadas.
- Tabulação pode ser usada.
- Palavras de comando não podem ser quebras em linhas.
- Um comando SQL é entrado no SQL prompt, e as linhas subseqüentes são numeradas. Este é chamado de SQL Buffer.
- Somente um comando pode ser corrente no buffer, e pode ser executado de várias maneiras:
  - usando ponto-e-vírgula (;) no fim da última cláusula;
  - usando (;) ou (/) na última linha do buffer;
  - usando (/) no SQL prompt;
  - usando R[un] comando no SQL prompt.

Exemplo:

```
Select * from emp;
```

```
Select  
*  
from  
emp  
;
```

```
Select *  
from emp;
```

## Bloco de Pesquisa Básico

Um comando SELECT recupera informações da tabela-base.

Um SELECT faz essencialmente a projeção das colunas a ser exibidas. O FROM especifica qual a tabela invocada.

```
Select deptno,ename,mgr from emp;
** selecione colunas da tabela **
```

As colunas são sempre separadas por vírgula (,).

```
Select * from emp;
```

O (\*) especifica que todas as colunas da tabela serão selecionadas.

## Expressões Aritméticas

As expressões aritméticas podem conter nomes de colunas, valores numéricos constantes e operadores aritméticos (+, -, \* (multiplicação), / (divisão)) sendo que esses operadores seguem as regras matemáticas de prioridade. Exemplo:

SQL> SELECT ENAME, SAL*12, COMM FROM EMP;		
ENAME	SAL*12	COMM
SMITH	9600	
ALLEN	19200	300
WARD	15000	500
JONES	35700	
MARTIN	15000	1400
BLAKE	34200	
CLARK	29400	
SCOTT	36000	
KING	60000	
TURNER	18000	0
ADAMS	13200	
JAMES	11400	
FORD	36000	
MILLER	15600	
14 rows selected.		

## Apelidos De Colunas

Observando o exemplo anterior, verificamos que o nome da coluna listada, foi "SAL\*12". Para mudar este cabeçalho devemos usar um "ALIAS" para a coluna.

Exemplo:

SQL> SELECT ENAME, (SAL*12) SALARIO_ANUAL, COMM FROM EMP;		
ENAME	SALARIO_ANUAL	COMM
SMITH	9600	
ALLEN	19200	300
WARD	15000	500
JONES	35700	
MARTIN	15000	1400
BLAKE	34200	
CLARK	29400	
SCOTT	36000	
KING	60000	
TURNER	18000	0
ADAMS	13200	
JAMES	11400	
FORD	36000	
MILLER	15600	
14 rows selected.		

## Operador De Concatenação

O operador de concatenação ( || ) permite listar duas colunas como uma só.

Exemplo:

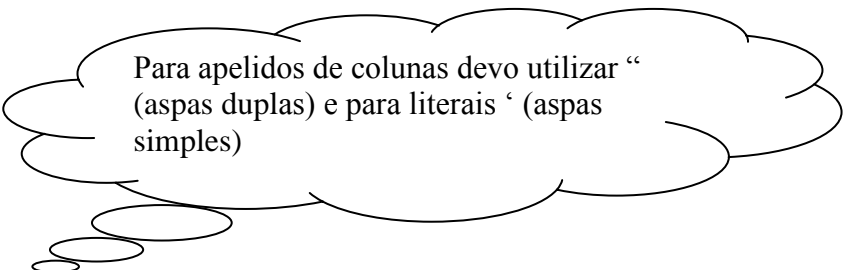
```
SQL> select empno || ename funcionario from emp;  
FUNCIONARIO  
7369SMITH  
7499ALLEN  
7521WARD  
7566JONES  
7654MARTIN  
7698BLAKE  
7782CLARK  
7788SCOTT  
7839KING  
7844TURNER  
7876ADAMS  
7900JAMES  
7902FORD  
7934MILLER  
14 rows selected.
```

## Literais

No exemplo anterior nós podemos colocar um literal (espaço, número ou um caracter) para uma melhor visualização. Para isso devemos colocar no comando SELECT o literal entre quotas ( ' ). Os literais numéricos não necessitam de quotas.

Exemplo:

```
SQL> select empno || ' ' || ename funcionario from emp;  
FUNCIONARIO  
7369-SMITH  
7499-ALLEN  
7521-WARD  
7566-JONES  
7654-MARTIN  
7698-BLAKE  
7782-CLARK  
7788-SCOTT  
7839-KING  
7844-TURNER  
7876-ADAMS  
7900-JAMES  
7902-FORD  
7934-MILLER  
14 rows selected.
```



Para apelidos de colunas devo utilizar “  
(aspas duplas) e para literais ‘ (aspas  
simples)

## Trabalhando Com Valores Nulos

É muito importante converter sempre uma coluna que será utilizada em uma expressão e que pode conter nulos em um valor não nulo através da função NVL. Toda operação com um valor nulo, terá como resultado um valor nulo.

Exemplo:

```
SQL> select (sal*12)+comm renda_anual from emp;  
RENDA_ANUAL
```

19500

15500

16400

18000

14 rows selected.

Os espaços em branco indicam a presença de valores nulos na expressão.

```
SQL> select (sal*12) + nvl(comm,0) renda_anual from emp;
```

RENDA\_ANUAL

9600

19500

15500

35700

16400

34200

29400

36000

60000

18000

13200

11400

36000

15600

14 rows selected.

## Outros Itens Do Select (Cláusulas)

### Prevenindo a Seleção de Linhas Duplicadas (DISTINCT)

Ao utilizarmos a cláusula DISTINCT no comando SELECT estaremos eliminando do resultado os valores duplicados das colunas.

Exemplo:

```
SQL> SELECT DISTINCT DEPTNO FROM EMP;  
DEPTNO  
10  
20  
30
```

Várias colunas podem ser especificadas após DISTINCT, e o resultado será a combinação de todas as selecionadas. O DISTINCT deve ser referenciado apenas uma vez e sempre após o SELECT.

Exemplo:

```
SQL> select distinct deptno, job from scott.emp;  
DEPTNO  JOB  
-----  
10 CLERK  
10 MANAGER  
10 PRESIDENT  
20 ANALYST  
20 CLERK  
20 MANAGER  
30 CLERK  
30 MANAGER  
30 SALESMAN  
9 rows selected.
```



### Ordenando a Saída do SELECT (ORDER BY)

A ordem das linhas do resultado de um SELECT normalmente é indefinida. A cláusula ORDER BY é utilizada para classificar as linhas do resultado. Esta cláusula deve ser sempre a última de um comando SELECT.

Exemplo:

```
SQL> SELECT ENAME, SAL FROM EMP ORDER BY ENAME;
ENAME      SAL
-----
ADAMS      1100
ALLEN      1600
BLAKE      2850
CLARK      2450
FORD       3000
JAMES      950
JONES      2975
KING       5000
MARTIN     1250
MILLER     1300
SCOTT      3000
SMITH      800
TURNER     1500
WARD       1250
14 rows selected.
```

O default é ordem ascendente, sendo ordenado por: valores numéricos, datas, caracteres alfabéticos maiúsculos e depois os minúsculos. **Os valores nulos são mostrados por último.**

Exemplo:

```
SQL> select ename, job, sal, comm from scott.emp order by comm, sal;
ENAME      JOB      SAL      COMM
-----
TURNER     SALESMAN    1500      0
ALLEN      SALESMAN    1600     300
WARD       SALESMAN    1250     500
MARTIN     SALESMAN    1250    1400
SMITH      CLERK        800
JAMES      CLERK        950
ADAMS      CLERK       1100
MILLER     CLERK       1300
CLARK      MANAGER     2450
BLAKE      MANAGER     2850
JONES      MANAGER     2975
SCOTT      ANALYST     3000
FORD       ANALYST     3000
KING       PRESIDENT   5000
14 rows selected.
```

### Agrupando Dados no SELECT (GROUP BY)

A cláusula GROUP BY pode ser utilizada para dividir as linhas da tabela em grupos menores. Funções de grupo devem ser usadas para retornar informações sumarizadas de cada grupo.

Exemplo:

```
SQL> select deptno, sum(sal) from emp
2 group by deptno;
DEPTNO SUM(SAL)
-----
10      8750
20     10875
30      9400
```

Toda vez que utilizar uma função de grupo em conjunto com outra coluna, obrigatoriamente a coluna deve fazer parte da cláusula group by.

Exemplo:

```
SQL> select job, sum(sal), sum(comm) from scott.emp
2> group by job;
JOB          SUM(SAL) SUM(COMM)
-----
ANALYST      6000
CLERK        4150
MANAGER      8275
PRESIDENT    5000
SALESMAN     5600      2200
5 rows selected.
```

Se a coluna tiver  
valores nulos devo  
utilizar a função NVL

### Restringindo Os Grupos De Dados No Select (Having)

A cláusula HAVING especifica que grupos você deseja mostrar, ou seja, restringe os grupos informando a condição para listar.

Exemplo:

```
SQL> select job, avg(sal) from emp
2  group by job
3  having avg(sal) = (select max(avg(sal)) from emp
4  group by job);
```

JOB	AVG(SAL)
PRESIDENT	5000

## Realizando restrições no SELECT (WHERE)

Após esta cláusula você coloca as restrições da seleção que você deseja fazer. Esta pode comparar valores em colunas, valores literais, expressões aritméticas ou funções. A cláusula WHERE conta com 3 elementos:

1. O nome da coluna;
2. O operador de comparação;
3. Um nome de coluna, uma constante ou lista de valores;

Os operadores de comparação podem ser divididos em operadores lógicos e operadores SQL.

Exemplo:

```
SQL> SELECT ENAME, SAL FROM EMP
2 WHERE DEPTNO = 10;
ENAME      SAL
-----
CLARK      2450
KING       5000
MILLER     1300
```

## Operadores Lógicos

= Igual a

```
SQL> SELECT ENAME, SAL FROM EMP
2 WHERE DEPTNO = 20;

ENAME      SAL
-----
SMITH      800
JONES      2975
SCOTT      3000
ADAMS      1100
FORD       3000

5 rows selected.
```

> **Maior que**

```
SQL> SELECT ENAME, SAL FROM EMP
2 WHERE DEPTNO > 20;
ENAME      SAL
-----
ALLEN      1600
WARD       1250
MARTIN     1250
BLAKE      2850
TURNER     1500
JAMES      950

6 rows selected.
```

>= **Maior ou igual a**

```
SQL>SELECT ENAME, SAL FROM EMP
2 WHERE DEPTNO >= 30;
ENAME      SAL
-----
ALLEN      1600
WARD       1250
MARTIN     1250
BLAKE      2850
TURNER     1500
JAMES      950

6 rows selected.
```

< **Menor que**

```
SQL> SELECT ENAME, SAL FROM EMP
2 WHERE DEPTNO < 20;

ENAME      SAL
-----
CLARK      2450
KING       5000
MILLER     1300
```

**<=**    **Menor ou igual a**

```
SQL> SELECT ENAME, SAL FROM EMP  
2  WHERE DEPTNO <= 20;
```

ENAME	SAL
SMITH	800
JONES	2975
CLARK	2450
SCOTT	3000
KING	5000
ADAMS	1100
FORD	3000
MILLER	1300

8 rows selected.

## Operadores SQL

### **BETWEEN .. AND**

Seleciona as informações constantes dentro da faixa estipulada pelo parametro inicial e o parametro final (inclusive).

Exemplo:

```
SQL> SELECT ENAME, SAL FROM EMP  
2 WHERE SAL BETWEEN 1000 AND 2000;
```

ENAME	SAL
ALLEN	1600
WARD	1250
MARTIN	1250
TURNER	1500
ADAMS	1100
MILLER	1300

6 rows selected.

**IN (list)**

Seleciona os valores constantes de acordo com a lista dos parametros definidos.

Exemplo:

```
SQL> SELECT ENAME, SAL, DEPTNO FROM EMP  
2 WHERE DEPTNO IN (10,20);
```

ENAME	SAL	DEPTNO
SMITH	800	20
JONES	2975	20
CLARK	2450	10
SCOTT	3000	20
KING	5000	10
ADAMS	1100	20
FORD	3000	20
MILLER	1300	10

8 rows selected.



**LIKE**

Seleciona os dados referentes ao parametro definido. Se utilizado com o coringa % será considerado todo o tamanho do texto e com qualquer letra. Se utilizado com \_ (sublinhado) será considerado somente uma posição por \_.

Exemplo:

```
SQL> SELECT ENAME FROM EMP WHERE ENAME LIKE 'S%';
ENAME
SMITH
SCOTT

SQL> SELECT ENAME FROM EMP WHERE ENAME LIKE '____';

no rows selected
--
-- Com tres sublinhados
--
SQL> SELECT ENAME FROM EMP WHERE ENAME LIKE '_____';

ENAME
WARD
KING
FORD
--
-- Com quatro sublinhados
--
```

---

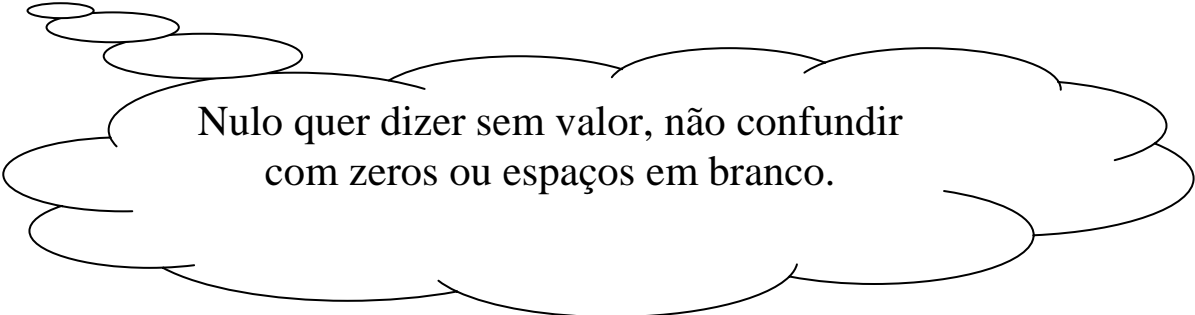
**IS NULL**

---

Seleciona os dados que tenham valor nulo.

Exemplo:

```
SQL> SELECT ENAME FROM EMP WHERE MGR IS NULL;  
ENAME  
KING
```



Nulo quer dizer sem valor, não confundir  
com zeros ou espaços em branco.

---

**Expressões De Negação**

---

!= ou <>	NÃO IGUAL A (TODOS O/S)
NOT colname =	NÃO IGUAL A
NOT colname >	NÃO MAIOR QUE
NOT BETWEEN	NÃO ENTRE DOIS VALORES DADOS
NOT IN	NÃO NA LISTA DE VALORES DADA
NOT LIKE	NÃO SEMELHANTE A
IS NOT NULL	NÃO NULO

## Substituição De Variáveis

### &

Para substituir valores em tempo de execução, coloque a variável precedida por (&).

```
SQL> select ename from emp where deptno = &depto;
Enter value for depto: 10
old 1: select ename from emp where deptno = &depto
new 1: select ename from emp where deptno = 10
ENAME
CLARK
KING
MILLER
```

### &&

Se a variável é prefixada por dois & comercial, o SQL\*Plus pedirá somente uma vez o valor da variável. A cada nova execução terá o mesmo valor.

```
SQL> select ename from emp where deptno = &&depto;
Enter value for depto: 10
old 1: select ename from emp where deptno = &depto
new 1: select ename from emp where deptno = 10
ENAME
CLARK
KING
MILLER
SQL> /
old 1: select ename from emp where deptno = &depto
new 1: select ename from emp where deptno = 10
ENAME
CLARK
KING
MILLER
```

**DEFINE**

Pode-se usar o comando DEF[INE] do SQL\*Plus para assinalar um conteúdo a uma variável. O valor definido na variável será referenciado pelo comando SELECT colocando-se o prefixo &. Para liberar as variáveis vazias ou não utilizadas usa-se o comando UNDEFINE.

```
SQL> DEFINE REM='SAL*12+NVL(COMM,0)'
SQL> SELECT ENAME, JOB, &REM
  2  FROM EMP
  3  ORDER BY &REM;
old  1: SELECT ENAME, JOB, &REM
new  1: SELECT ENAME, JOB, SAL*12+NVL(COMM,0)
old  3: ORDER BY &REM
new  3: ORDER BY SAL*12+NVL(COMM,0)
```

```
ENAME    JOB      SAL*12+NVL(COMM,0)
-----
```

SMITH	CLERK	9600
JAMES	CLERK	11400
ADAMS	CLERK	13200
WARD	SALESMAN	15500
MILLER	CLERK	15600
MARTIN	SALESMAN	16400
TURNER	SALESMAN	18000
ALLEN	SALESMAN	19500
CLARK	MANAGER	29400
BLAKE	MANAGER	34200
JONES	MANAGER	35700
SCOTT	ANALYST	36000
FORD	ANALYST	36000
KING	PRESIDENT	60000

```
14 rows selected.
```

```
SQL> UNDEFINE REM
SQL>
```

## Pesquisando Dados Com Múltiplas Condições

Os operadores AND e OR são usados para compor expressões lógicas.

Com o operador AND as duas condições que compõe a expressão lógica devem ser verdadeiras.

Com o operador OR apenas uma das duas condições tem que ser verdadeira.

```
SQL> select empno, ename, job, sal
2  from emp
3  where sal between 1000 and 2000
4  AND job = 'CLERK';
```

EMPNO	ENAME	JOB	SAL
7876	ADAMS	CLERK	1100
7934	MILLER	CLERK	1300

```
SQL> select empno, ename, job, sal
2  from emp
3  where sal between 1000 and 2000
4  OR job = 'CLERK';
```

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	800
7499	ALLEN	SALESMAN	1600
7521	WARD	SALESMAN	1250
7654	MARTIN	SALESMAN	1250
7844	TURNER	SALESMAN	1500
7876	ADAMS	CLERK	1100
7900	JAMES	CLERK	950
7934	MILLER	CLERK	1300

8 rows selected.

## Precedência dos Operadores

=, !=, <, >, <=, >=, BETWEEN...AND, IN, LIKE, IS NULL

NOT

AND

OR

## SUMARIO DO SELECT

SELECT	(DISTINCT) *, COLUNA [ALIAS] , ...
FROM	TABELA
WHERE	CONDIÇÃO
ORDER	BY (COLUNA, EXPRESSÃO) (ASC/DESC)

**SELECT:** Indica o que será mostrado.

**FROM:** Indica de onde serão visualizados os dados.

**WHERE:** Indica a condição para selecionar os dados.

**ORDER:** Indica a ordem de visualização dos dados.

**[DISTINCT]:** Todas as colunas.

**[\*,COLUNA [ALIAS],...]:** Cabeçalho da coluna.

**CONDIÇÃO:** Indica a condição de projeção dos dados da tabela.

**[ASC/DESC]:** Indica a ordem: ASCendente ou DESCendente.

# Funções

As funções são utilizadas para manipular dados. Recebe um ou mais argumentos como parâmetros e retorna sempre um valor. Os argumentos podem ser constantes, variáveis ou colunas. A sintaxe é:

```
NOME_FUNÇÃO (ARGUMENTO1, ARGUMENTO2,...)
```

## Funções Caracter De Linha Simples Com Retorno De Caracter

### LOWER

Força que o valor alfabético seja mostrado em letras minúsculas.

```
Select lower (dname) , lower ( 'TESTE SQL' ) ;  
lower (dname)  lower ( TESTE SQL )
```

```
-----  
research      teste sql  
sales         teste sql  
operations    teste sql  
accouting     teste sql
```

### UPPER

Força que o valor alfabético seja mostrado em letras maiúsculas.

```
Select ename from emp where ename = upper (&nome);  
Enter value for name : smith  
ENAME  
-----  
SMITH
```

## INITCAP

Força que somente a primeira letra de cada palavra seja maiúscula.

```
Select initcap (dname) , initcap (loc) from dept;
INITCAP (DNAME) INITCAP (LOC)
-----
Accounting      New York
Research        Dallas
Sales           Chicago
Operations      Boston
```

## LPAD

Preenche a ESQUERDA da coluna ou valor literal de 'n' posições com o caráter 'string'.

```
Select Lpad ( dname, 20, '*' ) , Lpad (dname, 20) from dept;
LPAD (DNAME, 20, '*')      LPAD (DNAME, 20)
-----
*****RESEARCH           RESEARCH
*****SALES               SALES
*****OPERATIONS         OPERATIONS
*****ACCOUNTING          ACCOUNTING
```

## RPAD

Preenche a DIREITA da coluna ou valor literal de 'n' posições com o caráter 'string'.

```
Select Rpad (dname, 20, '*' ) , Rpad (dname, 20) from dept;
RPAD (DNAME, 20, '*')      RPAD (DNAME, 20)
-----
RESEARCH*****           RESEARCH
SALES*****              SALES
OPERATIONS*****         OPERATIONS
ACCOUNTING*****          ACCOUNTING
```



### SUBSTR (col | valor, pos, n)

Retorna uma string da coluna ou valor começando na posição 'pos' e com o tamanho 'n'.

```
Select substr ( 'ORACLE', 2 ,4 ), substr (dname, 2 ) , substr (dname3, 5)
from dept ;
SUBSTR ( 'ORACLE',2,4 ) SUBSTR (DNAME,2) SUBSTR (DNAME,3,5 )
-----
RACL                ESEARCH                SEARC
RACL                ALES                LES
RACL                PERATIONS            ERATI
RACL                CCOUNTING                COUNT
```

### LTRIM (col | valor, 'char')

Remove começando da ESQUERDA, os caracteres ou combinação de caracteres especificados em 'char'.

```
Select dname, ltrim ( dname, 'A' ) , ltrim (dname, 'ASOP') from dept;
DNAME          LTRIM (DNAME, 'A')          LTRIM ( DNAME, 'ASOP')
-----
RESEARCH       RESEARCH                RESEARCH
SALES          SALES                  LES
OPERATIONS     OPERATIONS            ERATIONS
ACCOUNTING     CCOUNTING              CCOUNTING
```

### RTRIM (col | valor, 'char')

Remove, começando da DIREITA, os caracteres ou combinação de caracteres especificados em 'char'. Se 'char' não for especificado, remove brancos.

```
Select dname rtrim (dname, 'G'), rtrim (dname, 'GHS') from dept;
DNAME          RTRIM (DNAME, 'G')          RTRIM (DNAME, 'GHS')
-----
RESEACH        RESEARCH                RESEARC
SALES          SALES                  SALE
OPERATIONS     OPERATIONS            OPERATION
ACCOUNTING     ACCOUNTIN              ACCOUNTIN
```

### **SOUNDEX(col | valor)**

Retorna a representação fonética da palavra pertencente à coluna ou valor especificado. Permite também a comparação de palavras diferentes mas com som parecido.

```
Select ename , soundex (ename) from emp
where soundex (ename) = soundex ( 'FRED');
ENAME          SOUNDEX (ENAME)
-----
FORD           F630
```

### **TRANSLATE (col | valor, from, to)**

Substitui na coluna ou valor literal, os caracteres existentes em 'from' pelos existentes em 'to'. A substituição é feita um a um.

```
Select ename, translate ( ename, 'k', 't')
from emp where deptno = 10;
ENAME          TRANSLATE ( ENAME, 'K', 'T')
-----
CLARK          CLART
KING           TING
MILLER         MILLER
```

### **REPLACE(col | valor, string, string\_substituta)**

Substitui na coluna ou valor literal, uma string por outra string.

```
Select job, replace ( job , 'PRESIDENT', 'BOSS') SUBSTITUTO
JOB          SUBSTITUTO
-----
MANAGER      MANAGER
PRESIDENT    BOSS
CLERK        CLERK
```

## Funções Caracter De Linha Simples Com Retorno De Número

### **LENGTH (col | valor)**

Retorna o número de caracteres ou dígitos da coluna ou valor literal.

```
Select length (dname) , dname from dept;
LENGTH (DNAME)    DNAME
-----
      8      RESEARCH
      5      SALES
     10    OPERATIONS
     10    ACCOUNTING
```

### **INSTR (col | valor, 'string', pos, n)**

Mostra a posição da primeira ocorrência da 'string' na posição 'pos', se houver, e 'n' indica qual a ocorrência.

```
select dname, instr(dname,'A'), instr(dname,'C',1,2) from dept;
DNAME      INSTR(DNAME,'A') INSTR(DNAME,'C',1,2)
-----
ACCOUNTING          1          3
RESEARCH            5          0
SALES                2          0
OPERATIONS          5          0
```

## Funções Numéricas De Linha Simples Com Retorno De Número

### ROUND (col | valor, n)

Arredondamento de valores. Se 'n' é omitido, ignora casas decimais. Se 'n' é negativo, arredonda a esquerda da casa decimal.

```
select round(45.923,1) vl1, round(45.923) vl2, round(45.323,1) vl3,
round(45.323,-1) vl4, round(46.323,-2) vl5, round(sal/32,2) vl6
from emp where deptno = 10
```

VL1	VL2	VL3	VL4	VL5	VL6
45.9	46	45.3	50	0	156.25
45.9	46	45.3	50	0	76.56
45.9	46	45.3	50	0	40.63

### TRUNC(col | valor, n)

Trunca a coluna ou valor na 'n' casa decimal. Se 'n' é omitido, ignora as casas decimais. Se 'n' é negativo, trunca a esquerda da casa decimal.

```
select trunc(45.923,1) vl1, trunc(45.923) vl2, trunc(45.323,1) vl3,
trunc(45.323,-1) vl4, trunc(46.323,-2) vl5, trunc(sal/32,2) vl6
from emp where deptno = 10
```

VL1	VL2	VL3	VL4	VL5	VL6
45.9	45	45.3	40	0	156.25
45.9	45	45.3	40	0	76.56
45.9	45	45.3	40	0	40.62

### CEIL (col | valor)

Retorna o menor inteiro maior ou igual a coluna ou valor especificado.

```
select ceil (101.77) from dual;
CEIL(101.77)
102
```

**FLOOR (col | valor)**

Retorna o maior inteiro menor ou igual a coluna ou valor especificado.

```
select floor(101.77) from dual;  
FLOOR(101.77)  
101
```

**POWER (col | valor,n)**

Eleva a coluna ou valor à potência especificada em 'n'.

```
select power (3,3) from dual;  
POWER(3,3)  
27
```

**ABS (col | valor)**

Indica o valor absoluto da coluna ou valor especificado.

```
select (150-200), abs(150-200) from dual;  
(150-200) ABS(150-200)  
-----  
50          50
```

**MOD (valor1, valor2)**

Retorna o RESTO da divisão do valor1 pelo valor2.

```
select mod(1000,100), mod(1150,100) from dual;  
MOD(1000,100) MOD(1150,100)  
-----  
0           50
```

**SQRT (col | valor)**

Retorna a raiz quadrada da coluna ou valor. Se a coluna ou valor for menor que zero, nulo é retornado.

```
select sqrt(9) from dual;  
SQRT(9)  
3
```

**SIGN(col | valor)**

Retorna -1 se a coluna ou valor for um número negativo, retorna 0 se for zero e +1 se for positivo.

```
select sign(-20), sign(20), sign(0) from dual;  
SIGN(-20) SIGN(20) SIGN(0)
```

1	1	0
---	---	---

## Funções De Data

### **MONTHS\_BETWEEN(data1, data2)**

Retorna o número de meses entre duas datas. Se a data1 é posterior a data2, o resultado é positivo, caso contrário é negativo.

```
SELECT MONTHS_BETWEEN('01-JAN-98','01-MAR-98') DATA1,
MONTHS_BETWEEN('01-JAN-98','01-MAR-97') DATA2
FROM DUAL
DATA1  DATA2
-----
2      10
```

### **ROUND(data1, char)**

Retorna data1 para meia-noite do dia atual se a hora for menor que meio-dia, ou meia-noite do dia seguinte se a hora for maior que meio-dia.

Se 'char' for igual MONTH, então retorna a data do primeiro dia do mês de data1 se o dia pertencer a primeira quinzena caso contrario, retorna a data do primeiro dia do mês seguinte.

Se 'char' for igual a YEAR, então retorna a data do primeiro dia do primeiro mês do ano da data1, se o mês pertencer ao primeiro semestre, caso contrário, retorna a data do primeiro dia do primeiro mês do ano seguinte.

```
SELECT SYSDATE DATA1,
ROUND(SYSDATE,'MONTH') DATA2,
ROUND(SYSDATE,'YEAR') DATA3
FROM DUAL;
DATA1  DATA2  DATA3
-----
01-MAR-98 01-MAR-98 01-JAN-98
```

### **TRUNC(data1,char)**

Retorna a data do primeiro dia do mês contido em data1 quando 'char' = MONTH.

Se 'char'= YEAR, retorna o primeiro dia do ano contido em data1.

```
SELECT SYSDATE DATA1,
TRUNC(SYSDATE,'MONTH') DATA2,
TRUNC(SYSDATE,'YEAR') DATA3
FROM DUAL;
DATA1  DATA2  DATA3
-----
29-MAR-98 01-MAR-98 01-JAN-98
```

**ADD\_MONTHS(data,n)**

Adiciona 'n' número ao calendário de meses da 'data', 'n' deve ser inteiro e pode ser negativo.

```
SELECT SYSDATE DATA1,  
ADD_MONTHS(SYSDATE,2) DATA2,  
ADD_MONTHS(SYSDATE,-2) DATA3  
FROM DUAL  
DATA1 DATA2 DATA3  
-----  
29-MAR-98 29-MAY-98 29-JAN-98
```

**NEXT\_DAY(data1,char)**

Retorna o próximo dia da semana especificado em 'char', sendo que 'char' pode ser um número que representa o dia ou o dia da semana escrito em inglês.

```
SELECT SYSDATE DATA1, NEXT_DAY(SYSDATE,2) DATA2,  
NEXT_DAY(SYSDATE,'MONDAY') DATA3 FROM DUAL  
DATA1 DATA2 DATA3  
-----  
29-MAR-98 30-MAR-98 30-MAR-98
```

**LAST\_DAY(data1)**

Retorna a data do último dia do mês contido em 'data1'.

```
SELECT SYSDATE DATA1,  
LAST_DAY(SYSDATE) DATA2,  
LAST_DAY(SYSDATE-30) DATA3  
FROM DUAL  
DATA1 DATA2 DATA3  
-----  
29-MAR-98 31-MAR-98 28-FEB-98
```



## Funções De Conversão

### **TO\_CHAR (número | data, 'mascara edição')**

Converte um número ou uma data em formato caráter. Normalmente é mais usado para conversão de datas em um novo formato de saída.

```
SELECT TO_CHAR(SYSDATE,'DD/MM/YY') DIA,  
       TO_CHAR(SYSDATE,'HH:MI:SS') HORA  
FROM DUAL;  
DIA    HORA  
-----  
29/03/98 08:06:08
```

### **TO\_NUMBER (char)**

Converte o conteúdo de 'char' para número.

```
SELECT 1000+TO_NUMBER('1000') FROM DUAL;  
1000+TO_NUMBER('1000')  
-----  
2000
```

### **TO\_DATE (número | data, 'mascara edição')**

Converte um valor 'char' que representa uma data, em um valor data de acordo com a mascara de edição especificada. Se a máscara for omitida assume o formato 'DD-MON-YY'.

```
SELECT EMPNO, ENAME, HIREDATE FROM EMP  
WHERE HIREDATE = TO_DATE('04-JUN-60','DD-MON-YY')  
no rows selected
```

**FORMATOS DE DATA**

PICTURE	DESCRIÇÃO
SCC ou CC	Século; 'S' coloca prefixo 'BC' na data com '-';
YYYY ou SYYYY	Ano; 'S' coloca prefixo 'BC' na data com '-';
YYY ou YY ou Y	Ano com 3, 2, 1 dígito.
Y,YYY	Ano com vírgula na posição;
SYEAR ou YEAR	Nome do ano por extenso;
BC ou AD	Indicador de antes de Cristo ou depois de Cristo.
Q	Quarto de ano;
MM	Mês;
MONTH	Mês por extenso completo
MON	Mês por extenso abreviado;
WW ou W	Semana do ano ou do mês;
DDD ou DD ou D	Dia do ano, do mês, da semana;
DAY	Nome do dia completo;
DY	Nome do dia abreviado;
J	Dia juliano; O número de dias desde a data 31/12/31,4713 BC;
AM ou PM	Indicador do meridiano;
HH ou HH12	Horas do dia (1-12);
HH24	Horas do dia (1-24);
MI	Minuto;
SS	Segundos;
SSSSS	Segundos desde a meia-noite; (0-86399);
/., etc	Pontuação reproduzido no resultado;
"..."	String reproduzida no resultado;
Fm	Modalidade completa. Prefixado ao mês ou ao dia, suprime o estofamento em branco, saindo de um resultado do comprimento variável, FM suprimirá zero do título para o formato de DDTH. Não significativo com outros códigos. uma segunda ocorrência ' de FM ' gira o estofamento em branco sobre outra vez;
TH	Número ordinal;
SP	Nome do número; DDSP para FOUR
SPTH ou thsp	Nome do número ordinal;

## Outras Funções

### **GREATEST(col | valor1, col | valor2, ...)**

Retorna o maior valor entre a lista de valores comparados.

```
select greatest(100,200,150) from dual;  
GREATEST(100,200,150)  
200
```

### **VSIZE(col | valor)**

Retorna o número de bytes na representação interna do ORACLE para 'col | valor'.

```
select sal, vsize(sal) from emp  
where rownum < 5;  
SAL VSIZE(SAL)  
-----  
5000      2  
2850      3  
2450      3  
2975      3
```

### **DECODE(col | expressão, search1, result1, default)**

O valor de 'col | expressão' é comparado com o valor de 'search1' e retorna um resultado 'result1' se 'col | expressão' = 'search1'. Se for diferente retorna o valor de 'default'. Onde:

'col   expressão'	nome da coluna ou expressão a ser validada
'search1'	o primeiro valor a ser comparado
'result1'	o valor a ser retornado, caso 'col   expressão'='search1'
'default'	valor a ser retornado caso 'col   expressão' não encontre valor igual em 'search'
'search1' e 'result1' podem ser repetidos tantas vezes quanto for necessário.	

```
select ename, job, decode(RTRIM(job), 'CLERK', 'OPERARIO',
'MANAGER', 'CHEFE', 'INDEFINIDO') CARGO
FROM EMP
ENAME    JOB    CARGO
-----
KING     PRESIDENT INDEFINIDO
BLAKE    MANAGER  CHEFE
CLARK    MANAGER  CHEFE
JONES    MANAGER  CHEFE
MARTIN   SALESMAN INDEFINIDO
ALLEN    SALESMAN INDEFINIDO
TURNER   SALESMAN INDEFINIDO
JAMES    CLERK     OPERARIO
WARD     SALESMAN INDEFINIDO
FORD     ANALYST  INDEFINIDO
SMITH    CLERK     OPERARIO
SCOTT    ANALYST  INDEFINIDO
ADAMS    CLERK     OPERARIO
MILLER   CLERK     OPERARIO
14 rows selected.
```

### NVL(col | valor, valor1)

Converte um valor nulo para o 'valor1'. Sendo que 'col | valor' e 'valor1' devem ser do mesmo tipo.

```
SELECT COMM, NVL(COMM,0) FROM EMP;
COMM NVL(COMM,0)
-----
0
0
0
0
1400 1400
300 300
0 0
0
500 500
0
0
```

## Funções De Grupo

Funções de grupo operam com várias linhas de uma tabela. Retorna um resultado baseado no grupo de linhas. Por default, todas as linhas em uma tabela são tratadas como um grupo. A cláusula GROUP BY do comando SELECT é usada para dividir linhas em grupos menores.

### **AVG([distinct | all] n)**

Média do valor de 'n', ignora valores nulos.

```
SELECT AVG(SAL) FROM EMP;  
AVG(SAL)  
-----  
2073.2143
```

### **COUNT([distinct | all expr \*])**

Número de vezes que a expressão 'expr' aparece com valor que não seja nulo. O '\*' usado em COUNT, conta todas as linhas, incluindo duplas e nulas.

```
SELECT COUNT(*) FROM EMP;  
COUNT(*)  
14
```

### **MAX([distinct | all expr])**

Máximo valor para a expressão.

```
SELECT MAX(SAL) FROM EMP;  
MAX(SAL)  
5000
```

### **MIN([distinct | all expr])**

Minimo valor para a expressão.

```
SELECT MIN(SAL) FROM EMP  
MIN(SAL)  
-----  
800
```

**STDDEV([distinct | all n])**

Desvio padrão de 'n', ignorando valores nulos.

```
SELECT STDDEV(SAL) FROM EMP;  
STDDEV(SAL)  
1182.5032
```

**SUM([distinct | all n])**

Somatório de 'n'.

```
SELECT SUM(SAL) FROM EMP;  
SUM(SAL)  
29025
```

**Observações:**

DISTINCT faz agrupamento da função considerando somente valores distintos (não duplicados).

ALL faz agrupamento incluindo todos os valores, inclusive os duplicados. O default é ALL.

Todas as funções de grupo, com exceção de COUNT(\*), ignoram valores nulos.

NVL é utilizada para incluir valores nulos no agrupamento.

# Pseudo-Colunas

## SEQUENCES

### CURRVAL

Valor corrente da sequence.

### NEXTVAL

Próximo valor da sequence.

## SYSDATE

Retorna a data e o tempo corrente.

## LEVEL

Para cada linha selecionada por uma pesquisa hierárquica, o LEVEL retorna 1 para a linha raiz e vai acrescentando 1 a cada novo nível.

```
SELECT LEVEL, LPAD(' ', 2*(LEVEL-1)) || ename org_chart,  
empno, mgr, job FROM emp  
START WITH job = 'PRESIDENT'  
CONNECT BY PRIOR empno = mgr
```

LEVEL	ORG_CHART	EMPNO	MGR	JOB
1	KING	7839		PRESIDENT
2	BLAKE	7698	7839	MANAGER
3	MARTIN	7654	7698	SALESMAN
3	ALLEN	7499	7698	SALESMAN
3	TURNER	7844	7698	SALESMAN
3	JAMES	7900	7698	CLERK
3	WARD	7521	7698	SALESMAN
2	CLARK	7782	7839	MANAGER

## ROWID

Retorna o endereço da linha dentro do banco de dados em hexadecimal.

Esta cadeia hexadecimal é dividida em 3 partes:

- Bloco  
é uma cadeia hexadecimal que identifica o bloco dos dados do arquivo de dados de que contem a linha. O comprimento desta cadeia pode variar dependendo de seu sistema operacional.
- Linha  
é uma cadeia hexadecimal de quatro dígitos que identifica a linha no bloco dos dados. A primeira linha no bloco tem o número 0.
- Arquivo  
é uma cadeia hexadecimal que identifica o arquivo de base de dados que contem a linha. O primeiro arquivo de dados tem o número 1. O comprimento desta cadeia pode variar dependendo de seu sistema operacional.

```
SELECT ROWID, EMPNO FROM EMP;  
ROWID      EMPNO  
-----  
000000A8.0000.0002  7839  
000000A8.0001.0002  7698  
000000A8.0002.0002  7782  
000000A8.0003.0002  7566  
000000A8.0004.0002  7654  
000000A8.0005.0002  7499  
000000A8.0006.0002  784
```



## ROWNUM

Retorna o número de ordem da linha dentro da tabela. Use o ROWNUM para limitar o número de linhas que retornara de uma pesquisa, conforme exemplo:

```
SELECT * FROM emp WHERE ROWNUM < 10
EMPNO ENAME  JOB      MGR HIREDATE    SAL  COMM  DEPTNO
-----
7839 KING    PRESIDENT 17-NOV-81  5000      10
7698 BLAKE   MANAGER   7839 01-MAY-81  2850      30
7782 CLARK   MANAGER   7839 09-JUN-81  2450      10
7566 JONES   MANAGER   7839 02-APR-81  2975      20
7654 MARTIN  SALESMAN  7698 28-SEP-81  1250    1400    30
7499 ALLEN   SALESMAN  7698 20-FEB-81  1600    300     30
7844 TURNER  SALESMAN  7698 08-SEP-81  1500      0       30
7900 JAMES   CLERK     7698 03-DEC-81   950      30
7521 WARD    SALESMAN  7698 22-FEB-81  1250    500     30
```

9 rows selected.

```
SELECT ROWNUM, EMPNO FROM EMP
ROWNUM  EMPNO
-----
```

```
1  7839
2  7698
3  7782
4  7566
5  7654
6  7499
7  7844
8  7900
9  7521
10 7902
11 7369
12 7788
13 7876
14 7934
```

14 rows selected.

Pode-se usar também ROWNUM para assinalar à cada linha da tabela um valor único, conforme exemplo:

```
UPDATE tabx SET col1 = ROWNUM
```

### DUAL

É uma tabela do usuário SYS e pode ser acessada por todos os usuários. Ela contém uma coluna, DUMMY, e uma linha com valor 'X'. A DUAL é normalmente usada quando se deseja retornar um único valor, ou um valor constante, ou uma pseudo\_coluna, ou uma expressão que não deriva de nenhuma tabela.

### USER

Retorna o nome do usuário corrente conectado ao Oracle.

### UID

Retorna o numero de identificação assinalado ao usuário.

# Junção

A junção é usada quando o SQL requer informações de mais de uma tabela. Linhas de uma tabela devem ser juntas a linhas de outras tabelas de acordo com o valor existente nas colunas correspondentes. Há dois tipos de junção.

- EQUI-JOIN
- NON-EQUI-JOIN

## Equi-Join

Quando o relacionamento entre duas tabelas se faz através de uma coluna comum entre ambas e que o operador de comparação (=) é usado. Uma condição de junção é especificada na cláusula WHERE, conforme a sintaxe abaixo.

```
SELECT coluna (s) FROM tabelas WHERE condição de junção.
```

```
SQL> select ename, job, dname from emp, dept
```

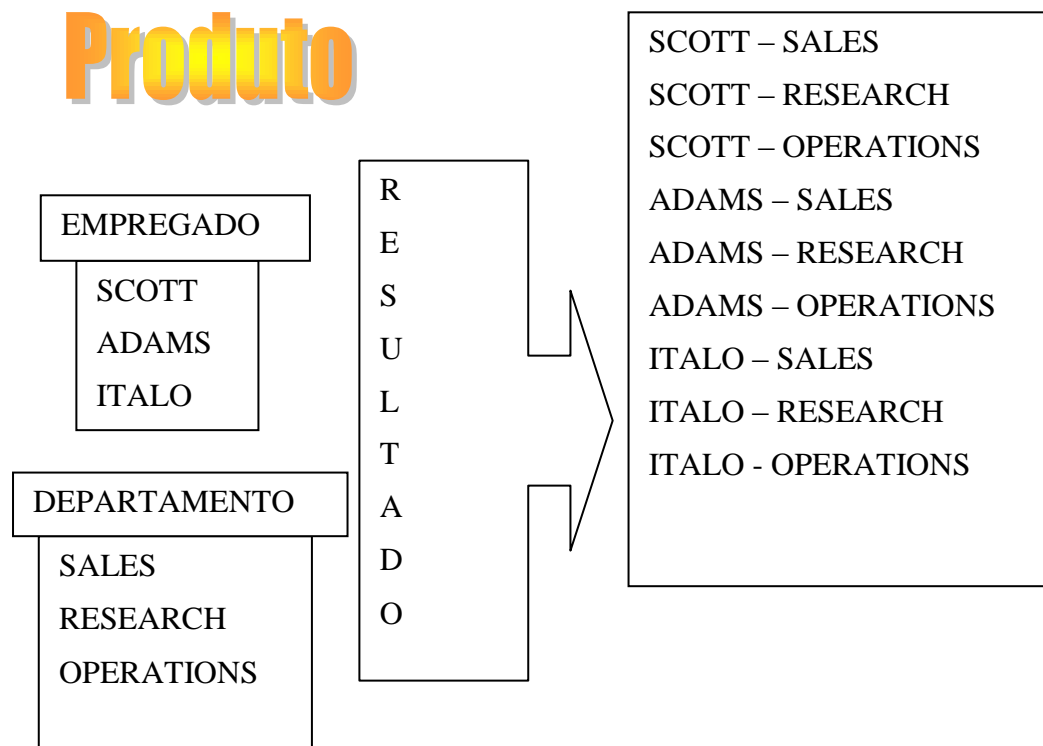
```
2 where emp.deptno = dept.deptno;
```

```
ENAME      JOB      DNAME
```

```
-----
```

CLARK	MANAGER	ACCOUNTING
KING	PRESIDENT	ACCOUNTING
MILLER	CLERK	ACCOUNTING
SMITH	CLERK	RESEARCH
ADAMS	CLERK	RESEARCH
FORD	ANALYST	RESEARCH
SCOTT	ANALYST	RESEARCH
JONES	MANAGER	RESEARCH
ALLEN	SALESMAN	SALES
BLAKE	MANAGER	SALES
MARTIN	SALESMAN	SALES
JAMES	CLERK	SALES
TURNER	SALESMAN	SALES
WARD	SALESMAN	SALES

```
14 rows selected.
```



### Uso De Aliases Para Nomes De Tabelas

Podemos utilizar um título temporário (alias) para nomear as tabelas na cláusula FROM. Isto evita que se repita o nome da tabela para cada coluna mencionada.

```
SQL> select e.ename, d.deptno, d.dname from emp e, dept d
2  where e.deptno = d.deptno
3  order by d.deptno;
```

ENAME	DEPTNO	DNAME
CLARK	10	ACCOUNTING
KING	10	ACCOUNTING
MILLER	10	ACCOUNTING
SMITH	20	RESEARCH
ADAMS	20	RESEARCH
FORD	20	RESEARCH
SCOTT	20	RESEARCH
JONES	20	RESEARCH
ALLEN	30	SALES
BLAKE	30	SALES
MARTIN	30	SALES
JAMES	30	SALES
TURNER	30	SALES
WARD	30	SALES

14 rows selected.

## Non-Equi-Join

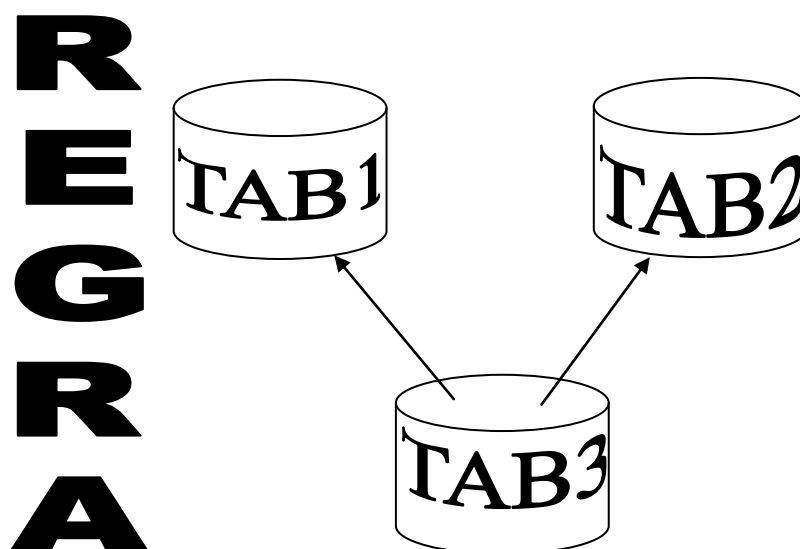
Quando não existe uma coluna correspondente direta entre as tabelas, e na cláusula WHERE usa-se o operador de comparação diferente de ( $\neq$ ).

**REGRA:** O número de tabelas menos um é igual ao número mínimo de condições de junção.

```
SQL> select e.ename, e.sal, s.grade  
2   from emp e, salgrade s  
3  where e.sal between s.losal and s.hisal;
```

ENAME	SAL	GRADE
SMITH	800	1
ADAMS	1100	1
JAMES	950	1
WARD	1250	2
MARTIN	1250	2
MILLER	1300	2
ALLEN	1600	3
TURNER	1500	3
JONES	2975	4
BLAKE	2850	4
CLARK	2450	4
SCOTT	3000	4
FORD	3000	4
KING	5000	5

14 rows selected.



# Outro métodos de junção

## Outer Join

Se uma linha não satisfaz uma condição de join, ela não aparece no resultado da pesquisa. Para que esta linha apareça no resultado, usa-se OUTER JOIN, que consiste em colocar o sinal de mais entre parênteses (+) no lado da tabela deficiente da igualdade.

```
SQL> select e.ename, d.deptno, d.dname  
2  from emp e, dept d  
3  where e.deptno (+) = d.deptno  
4    and d.deptno in(30,40);
```

ENAME	DEPTNO	DNAME
ALLEN	30	SALES
BLAKE	30	SALES
MARTIN	30	SALES
JAMES	30	SALES
TURNER	30	SALES
WARD	30	SALES
	40	OPERATIONS

7 rows selected.

O operador OUTER JOIN pode somente aparecer em um dos lados da expressão, e sempre do lado que a informação esta faltando.

## Join Reflexivo

Este tipo de JOIN é utilizado para agrupar informações oriundas de uma linha da tabela com informações que vêm de outras linhas da mesma tabela. Tipo AUTO-RELACIONAMENTO.

```
SQL> select e.ename, m.ename from emp e, emp m
```

```
2  where e.mgr = m.empno;
```

ENAME	ENAME
-------	-------

-----	-----
-------	-------

SCOTT	JONES
-------	-------

FORD	JONES
------	-------

ALLEN	BLAKE
-------	-------

WARD	BLAKE
------	-------

JAMES	BLAKE
-------	-------

TURNER	LAKE
--------	------

MARTIN	BLAKE
--------	-------

MILLER	CLARK
--------	-------

ADAMS	SCOTT
-------	-------

JONES	KING
-------	------

CLARK	KING
-------	------

BLAKE	KING
-------	------

SMITH	FORD
-------	------

13 rows selected.



# Operadores de Conjunto

São usados na construção de pesquisa com diferentes tabelas. É a combinação de resultados de dois ou mais comandos SELECT em um único resultado. Agrupamento vertical de tabelas.

## Union

Se uma pesquisa retorna 'n' linha e uma segunda pesquisa retorna 'm' linha, a união das duas retornará 'n+m' linhas, menos as duplicadas.

```
SQL> select ename, 'sal' , sal from emp
2 union
3 select ename, 'com' , comm from emp where comm is not null;
ENAME      'SA'    SAL
-----
ADAMS      sal    1100
ALLEN      com     300
ALLEN      sal    1600
BLAKE      sal    2850
CLARK      sal    2450
FORD       sal    3000
JAMES      sal     950
JONES      sal    2975
KING       sal    5000
MARTIN     com    1400
MARTIN     sal    1250
MILLER     sal    1300
SCOTT      sal    3000
SMITH      sal     800
TURNER     com       0
TURNER     sal    1500
WARD       com     500
WARD       sal    1250

18 rows selected.
```

## Intersect

Retorna somente linhas recuperadas em ambas as pesquisas (interseção).

```
SQL> select job from emp where deptno = 10
2 intersect
3* select job from emp where deptno = 30;

JOB
CLERK
MANAGER
```

## Minus

Retorna todas as linhas recuperadas pela primeira coluna que não estão na segunda.

```
SQL> select job from emp where deptno = 10
2 minus
3* select job from emp where deptno = 30;

JOB
PRESIDENT
```

## OBSERVAÇÕES

Nas pesquisas que utilizam operadores de conjunto convém observar o seguinte:

1. Todos os SELECT devem ter o mesmo número de colunas selecionadas, e seus tipos devem ser um a um, idênticos. As conversões eventuais devem ser feitas no interior do SELECT com a ajuda das funções de conversão (TO\_CHAR, TO\_DATE, TO\_NUMBER). **NÃO** se pode utilizar o tipo de dado LONG.
2. As duplicidades são eliminadas, o DISTINCT é implícito.
3. Os nomes das colunas (TÍTULOS) são os do primeiro SELECT.
4. A largura das colunas é o maior valor entre todas as linhas selecionadas pelo SELECT.
5. Se um ORDER BY é utilizado, ele deve fazer referência ao número da coluna e não ao seu nome.

# Subqueries

Subquery é um comando SELECT dentro de outro comando SELECT e que retorna valores intermediários.

## Subquery Com Retorno De Uma Única Linha

Quando a subquery retornar um única linha, devemos utilizar o operador de comparação igual (=).

```
SELECT coluna1, coluna2 FROM tabela
WHERE  coluna = ( SELECT coluna FROM tabela WHERE condição);
```

Exemplo:

```
SQL> select ename, job, sal from emp
2  where sal = (select min(sal) from emp);
```

ENAME	JOB	SAL
-----	-----	-----
SMITH	CLERK	800

## Subquery Com Retorno De Mais De Uma Linha

Quando o SELECT retorna mais de uma linha, devemos utilizar os operadores de comparação IN, ANY e ALL nas cláusulas WHERE e HAVING, em conjunção com os operadores lógicos (=, <>, <, >, >=, <=).

```
SQL> select ename, sal, deptno from emp
2  where sal IN (select min(sal) from emp group by deptno);
```

ENAME	SAL	DEPTNO
-----	-----	-----
SMITH	800	20
JAMES	950	30
MILLER	1300	10

ANY compara o valor com CADA valor retornado pela subquery.

Quando se usa ANY, a cláusula DISTINCT é geralmente utilizada para prevenir linhas duplicadas.

ALL compara o valor com TODOS os valores retornados pela subquery.

HAVING pode ser usado, em substituição à cláusula WHERE, para comparar itens agrupados. Neste caso se pode usar a cláusula GROUP BY, no SELECT externo para agregar o resultado.

**ANY**

```
SQL> select ename, sal, deptno from emp
2  where sal > ANY (select distinct sal from emp
3      where deptno = 30) order by sal desc;
```

ENAME	SAL	DEPTNO
-------	-----	--------

KING	5000	10
SCOTT	3000	20
FORD	3000	20
JONES	2975	20
BLAKE	2850	30
CLARK	2450	10
ALLEN	1600	30
TURNER	1500	30
MILLER	1300	10
WARD	1250	30
MARTIN	1250	30
ADAMS	1100	20

12 rows selected.

**ALL**

```
SQL> select ename, sal, job, deptno from emp
2  where sal > ALL (select distinct sal from emp
3      where deptno = 30) order by sal desc;
```

ENAME	SAL	JOB	DEPTNO
-------	-----	-----	--------

KING	5000	PRESIDENT	10
SCOTT	3000	ANALYST	20
FORD	3000	ANALYST	20
JONES	2975	MANAGER	20

## HAVING

```
SQL> select deptno, avg(sal) from emp
2  HAVING avg(sal) > (select avg(sal) from emp
3      where deptno = 30)
4  group by deptno;
DEPTNO  AVG(SAL)
-----
10 2916.6667
20  2175
```

## EXIST

```
SQL> select empno, ename, deptno from emp
2      where not exists(select deptno from dept
3*                      where dept.deptno = emp.deptno);

no rows selected
```

```
SQL> select deptno, dname from dept d
2      where not exists (select 1 from emp e
3                      where e.deptno = d.deptno);

DEPTNO DNAME
-----
40 OPERATIONS
```

## Comandos D.D.L.

São comandos de definição de estrutura de dados necessários a realização das seguintes tarefas:

- criação, alteração, e exclusão dos objetos do banco.
- Concessão e retirada de concessão de privilégios.
- Estabelecimento de opções de auditoria.
- Adição de comentários ao banco.

O Oracle realiza um COMMIT implícito da transação corrente sempre antes e depois de cada comando DDL.

COMANDO	DESCRIÇÃO
Alter cluster	Modifica as características de armazenamento ou aloca extensões ao cluster.
Alter database	Altera o banco de dados
Alter function	Alter a função
Alter index	Alter o índice
Alter package	Altera o pacote
Alter procedure	Altera a procedure
Alter profile	Altera um profile
Alter resource cost	Especifica a formula para calcular o custo dos recursos usados na sessão
Alter role	Altera um grupo de privilégios
Alter rollback segment	Altera um segmento de volta
Alter sequence	Altera uma sequencia
Alter snapshot	Altera uma visão remota
Alter snapshot log	Altera o log da visão remota
Alter table	Altera a tabela
Alter tablespace	Altera o espaço de trabalho lógico
Alter trigger	Altera o gatilho
Alter user	Altera o usuário
Alter view	Altera a visão
Analyze	Coleta estatísticas de performance da tabela, cluster ou índice.

Audit	Realiza uma auditoria no sistema.
Comment	Adiciona comentários a estrutura do banco.
Create cluster	Cria cluster de dados
Create controlfile	Cria o arquivo de controle do banco
Create database	Cria o banco de dados
Create database link	Cria uma ligação entre bancos
Create function	Cria uma função
Create index	Cria um índice para uma tabela
Create package	Cria a parte de definições do pacotes de rotinas
Create package body	Cria a parte procedural do pacote de rotinas
Create procedure	Cria uma procedure
Create profile	Cria um profile para usuários
Create role	Cria um grupo de privilégios
Create rollback Segment	Cria uma segmento de retorno
Create schema	Cria um esquema de trabalho
Create sequence	Cria uma seqüência de números
Create snapshot	Cria um retrato de uma tabela
Create snapshot log	Cria um log para um retrato
Create synonym	Cria um sinônimo para um objeto
Create table	Cria tabelas
Create tablespace	Cria espaços de trabalho
Create trigger	Cria gatilhos
Create user	Cria usuários
Create view	Cria visões de tabelas
Drop cluster	Apaga um cluster
Drop database link	Apaga uma ligação entre bancos remotos
Drop function	Apaga uma função
Drop index	Apaga um índice

Drop package	Apaga um pacote
Drop procedure	Apaga uma procedure
Drop profile	Apaga um profile
Drop role	Apaga um grupo de privilégios
Drop rollback segment	Apaga um segmento de rollback
Drop sequence	Apaga uma seqüência de números
Drop snapshot	Apaga um retrato
Drop snapshot log	Apaga o log do retrato
Drop synonym	Apaga o sinônimo
Drop table	Apaga uma tabela
Drop tablespace	Apaga uma area de trabalho
Drop trigger	Apaga um gatilho
Drop user	Apaga um usuário
Drop view	Apaga uma visão
Grant	Concede privilégios a um usuário
Noaudit	Não realiza auditoria
Rename	Renomeia um objeto
Revoke	Revoga os privilégios concedidos
Truncate	Apaga um objeto



## Criação De Tabelas

Comando para criar tabelas para armazenamento de informações do usuário dentro do banco.

Sintaxe:

```
CREATE TABLE [schema.]table
( { column datatype [DEFAULT expr] [column_constraint] | table_constraint}
[, { column datatype [DEFAULT expr] [column_constraint] | table_constraint} ]...)
[ [PCTFREE integer] [PCTUSED integer]
[INITTRANS integer] [MAXTRANS integer]
[TABLESPACE tablespace]
[STORAGE ( [INITIAL integer [K|M] ]
           [NEXT integer [K|M] ]
           [PCTINCREASE integer]
           [MINEXTENTS integer]
           [MAXEXTENTS integer]
           [FREELIST GROUPS integer]
           [FREELISTS integer] )]
[ PARALLEL ( [ DEGREE { integer | DEFAULT } ]
[ INSTANCES { integer | DEFAULT } ] )
| NOPARALLEL ]
[ CACHE | NOCACHE ]
| [CLUSTER cluster (column [, column]...)] ]
[ ENABLE { {UNIQUE (column [, column] ...)
| PRIMARY KEY | CONSTRAINT constraint}
[USING INDEX [INITTRANS integer [MAXTRANS integer]
           [TABLESPACE tablespace [STORAGE storage_clause]
           [PCTFREE integer] [PARALLEL [integer] | NOPARALLEL]
           [EXCEPTIONS INTO [schema.]table ] | ALL TRIGGERS }
| DISABLE { { UNIQUE (column [, column] ...)
| PRIMARY KEY | CONSTRAINT constraint }
[CASCADE] | ALL TRIGGERS }} ...
[AS subquery]
```

Criação padrão de tabelas.

Exemplo:

```
CREATE TABLE Empresa
(Empresa      NUMBER(3) NOT NULL,
 Razao_Social VARCHAR2(40) NOT NULL,
 Endereco     VARCHAR2(40) NOT NULL,
 Bairro       VARCHAR2(20) NULL,
 Cidade       VARCHAR2(20) NULL,
 Estado       VARCHAR2(2) NOT NULL,
 Cep          NUMBER(8) NULL,
 Empresa_Mae  NUMBER(3) NULL
);
```

## TIPOS DE DADOS

VARCHAR2(size)	Alfanumerico com 2000 bytes.
NUMBER(p,s)	Numerico com $1.0 \times 10^{-129}$ a $9.99 \times 10^{124}$ ou 38 dígitos.
LONG	Similar ao varchar2, utilizado para armazenar textos longos. Até 2 Gb.
DATE	Data do periodo de 4712 AC ate 4712 DC. Armazena também dados do tempo (HH:MI:SS).
LONG RAW	Utilizado para armazenar imagens, som, suporta até 2 Gb.
CHAR(size)	Alfanumerico de até 255 bytes.

## CONSTRAINT PARAMETROS

CONSTRAINT [NOME]	Declara o nome da obrigação.
NULL/NOT NULL	Especifica se cada coluna deverá ser obrigatoria (preenchimento) ou não.
UNIQUE	Especifica a coluna utilizada para identificar valores distintos para cada linha.
PRIMARY KEY	Especifica a coluna utilizada para identificar cada linha como única.
FOREIGN KEY (column) / REFERENCES (user.table(cols))	Identifica a coluna que é chave estrangeira e referencia a chave primaria da outra tabela.
CHECK	Especifica a condição de preenchimento da coluna.

Criação de tabelas com especificação de regras definidas.

```
CREATE TABLE Empresa
(Empresa      NUMBER(3) NOT NULL PRIMARY KEY CONSTRAINT PK_EMPRESA,
 Razao_Social VARCHAR2(40) NOT NULL CHECK (RAZAO_SOCIAL = UPPER(RAZAO_SOCIAL)),
 Endereco     VARCHAR2(40) NOT NULL,
 Bairro       VARCHAR2(20) ,
 Cidade       VARCHAR2(20),
 Estado       VARCHAR2(2) NOT NULL,
 Cep          NUMBER(8),
 CGC          VARCHAR2(18) NOT NULL UNIQUE CONSTRAINT UK_EMPRESA,
 CGF          VARCHAR2(14) NOT NULL,
 Empresa_Mae  NUMBER(3) NULL REFERENCES EMPRESA(EMPRESA) CONSTRAINT FK_EMPRESA_MAE);
```

Criação de tabela a partir de outro esquema (usuário) com a cópia dos dados.

```
CREATE TABLE DEPTOS AS SELECT * FROM SCOTT.DEPT;
```

Para criar com os campos determinados, coloque o nome das colunas no select e caso deseje alterar os nomes coloque também os alias nas colunas que desejar renomear.

```
CREATE TABLE DEPTOS AS SELECT DEPTNO CODEPT, DNAME NOME FROM SCOTT.DEPT;
```

Para incluir a chave primaria.

Exemplo:

```
ALTER TABLE Empresa
ADD ( CONSTRAINT PK_EMPRESA
PRIMARY KEY (Empresa) );
```

Para incluir a chave estrangeira na tabela.

Exemplo:

```
ALTER TABLE Empresa
ADD ( CONSTRAINT FK_EMPRESA
FOREIGN KEY (Empresa_Mae)
REFERENCES Empresa ) ;
```

Para incluir uma obrigação na tabela, regra de preenchimento para as colunas.

Exemplo:

```
ALTER TABLE EMPRESA
ADD CONSTRAINT CK_EMPRESA01
CHECK (RAZAO_SOCIAL = UPPER(RAZAO_SOCIAL)) ;

ALTER TABLE EMPRESA
ADD CONSTRAINT CK_EMPRESA02
CHECK (ENDERECO = UPPER(ENDERECO)) ;

ALTER TABLE EMPRESA
ADD CONSTRAINT CK_EMPRESA03
CHECK (CIDADE = UPPER(CIDADE)) ;

ALTER TABLE EMPRESA
ADD CONSTRAINT CK_EMPRESA04
CHECK (BAIRRO = UPPER(BAIRRO)) ;

ALTER TABLE EMPRESA
ADD CONSTRAINT CK_EMPRESA05
CHECK (ESTADO IN
('AC','AL','AP','AM','BA','CE','DF','ES','GO','MA','MT','MS','MG','PA','PB','PR','PE','PI','RJ','RN','RS','RO','RR','SC','SP','SE','TO')) ;
```

Para apagar uma tabela com todos os objetos relacionados.

Exemplo:

```
DROP TABLE Empresa cascade constraints;
```

Para colocar comentários para uma tabela.

Exemplo:

```
COMMENT ON TABLE Empresa IS 'TABELA DE EMPRESAS';
```

Para colocar comentários para uma coluna.

Exemplo:

```
COMMENT ON COLUMN EMPRESA.CGC IS 'CADASTRO GERAL DE CONTRIBUINTES';
```

Para renomear uma tabela.

Exemplo:

```
RENAME old TO new;  
RENAME FUNCIONARIO TO FUNC;
```

# **DATA DICTIONARY**

O dicionário de dados do Oracle é composto pôr tabelas e visões.

As tabelas do dicionário não devem ser manipuladas diretamente pôr sua complexa compreensão. Elas são criadas no momento que se cria um banco de dados com o comando SQL "CREATE DATABASE".

O dicionário disponibiliza para os usuários visões simplificadas dos seus dados. Estas visões são classificadas em três classes de prefixos:

USER\_XXXX

Fornece informações sobre todos os objeto do usuário.

ALL\_XXXX

Fornece informações sobre todos os objetos que o usuário tem direito de acesso.

DBA\_XXXX

Fornece informações sobre todo o banco.

As únicas visões que não obedecem a esta regra são:

DICTIONARY	Informa todos os objetos que o usuário tem acesso.
DICT_COLUMNS	Informa as colunas dos objetos que o usuário tem acesso.
CONSTRAINT_DEF	Informa todas as definições de constraint das tabelas do usuário.
CONSTRAINT_COLUMNS	Informa todas as colunas das constraints das tabelas do usuário.

VISÃO	DESCRIÇÃO
2PC_NEIGHBORS(DBA_)	Informa sobre conexões E/S para transações distribuídas pendentes.
2PC_PENDING(DBA_)	Informa sobre transações distribuídas pendentes.
AUDIT_EXISTS(DBA_)	Entradas de trilha de auditoria pôr conta do Comando AUDIT EXISTS
AUDIT_OBJECT(DBA_,USER_)	Entradas de trilhas de auditoria para objetos do banco de dados
AUDIT_SESSION(DBA_,USER_)	Entradas de trilhas de auditoria concernentes a sessões do banco de dados
AUDIT_STATEMENT(DBA_,USER_)	Entradas de trilhas de auditoria para instruções auditadas
AUDIT_TRAIL(DBA_,USER_)	Conjunto de todas as entradas de trilha de auditoria
BLOCKERS(DBA_)	Sessões cujos bloqueios estão impedindo outras tran-

	sações de executar o trabalho. Veja visão relacionada DBA_WAITERS
CATALOG(ALL_,DBA_,USER_,(CAT))	Informações sobre tabelas do banco de dados, visões, sinônimos e seqüências
CLU_COLUMNS(DBA_,USER_)	Relação de colunas de tabela para agrupar chaves
CLUSTER(DBA_,USER_ (CLU))	Informações sobre clusters indexados e de prova do banco de dados
COL_COMMENTS(ALL_,DBA_,USER_)	Comentários para colunas de tabelas e visões
COL_PRIVS(ALL_,DBA_,USER_)	Informações sobre concessões em colunas específicas
COL_PRIVS_MADE (ALL_,USER_)	Informações sobre concessões em colunas específicas
COL_PRIVS_RECD (ALL_,USER_)	Informações sobre concessões recebidas em colunas específicas
CONS_COLUMNS (ALL_,DBA_,USER_)	Informações sobre colunas envolvidas em restrições de integridade
CONSTRAINTS (ALL_,DBA_,USER_)	Informações sobre restrições de integridade no banco de dados
DATA_FILES (DBA_)	Informações sobre os arquivos de dados do banco de dados
DB_LINKS (ALL_,DBA_,USER_)	Informações sobre associações de bancos de dados no banco de dados
DDL_LOCKS (DBA_)	Informações sobre bloqueios usados pôr conta de operações DDL
DEF_AUDIT_OPTS (ALL_)	Informações sobre opções default de auditoria de objetos
DEPENDENCIES (ALL_,DBA_,USER_)	Informações sobre dependências de objetos no banco de dados
DML_LOCKS (DBA_)	Informações sobre bloqueios de DML no servidor
ERROS (ALL_,DBA_,USER_)	Informações sobre erros de compilação detectados em procedimentos, funções, especificações e corpos de pacotes no banco de dados
EXP_FILES (DBA_)	Descrição dos arquivos exportados
EXP_OBJECTS (DBA_)	Informações sobre objetos que tenham sido exportados incrementalmente com utilitário Export
EXP_VERSION (DBA_)	Numero da versão da ultima sessão do utilitário Export
EXTENTS (DBA_,USER_)	Informações sobre extensões para objetos no banco de dados
FREE_SPACE (DBA_,USER_)	Informações sobre extensões disponíveis nas tablespaces de um banco de dados

IND_COLUMNS (ALL_, DBA_, USER_)	Informações sobre as colunas que correspondem a índices do banco de dados
INDEXES (ALL_, DBA_, USER_)	Informações sobre índices no banco de dados
LABELS (ALL_)	Informações sobre rótulos do sistema; útil somente com Trusted Oracle7
LOCKS (DBA_)	Informações sobre todos os bloqueios DDL e DML no servidor do banco de dados
MOUNTED_DBS (ALL_)	Informações sobre todos os bancos de dados montados; útil somente com Trusted Oracle7
OBJ_AUDIT_OPTS (DBA_, USER_)	Informações sobre opções de auditoria definidas para os objetos do banco de dados
OBJECT (ALL_, DBA_, USER_ (OBJ))	Informações sobre objetos de banco de dados no banco de dados
OBJECT_SIZE (DBA_, USER_)	Informações de tamanho para todos os procedimentos, funções, especificações e corpos de pacotes no banco de dados
PRIV_AUDIT_OPTS (DBA_)	Informações sobre auditoria para privilégios
PROFILES ( DBA_ )	Informações sobre perfis de limites de recursos em um banco de dados
RESOURCE_LIMITS ( USER_ )	Informações sobre limites de recursos para sessão atual de banco de dados
ROLES_PRIVS ( DBA_, USER_ )	Informações sobre personagens concedidos a um usuário
ROLES ( DBA_ )	Informações sobre personagens no banco de dados
ROLLBACK_SEGS ( DBA_ )	Informações sobre segmentos reconstitutivos no banco de dados
SEGMENTS ( DBA_, USER_ )	Informações sobre segmentos no banco de dados
SEQUENCES ( ALL_, DBA_, USER_ (SEQ))	Informações sobre seqüências no banco de dados
SNAPSHOT_LOGS ( DBA_ , USER_ )	Informações sobre registros de instantâneos no banco de dados
SNAPSHOTS ( ALL_, DBA_, USER_ )	Informações sobre instantâneos no banco de dados
SOURCE ( ALL_, DBA_, USER_ )	Código – fonte de procedimentos, funções, especificações e corpos de pacotes no banco de dados
STMT_AUDIT_OPTS ( DBA_ )	Informações sobre opções de auditoria definidas para instruções
SYNONYMS ( ALL_, DBA_, USER_ (SYN))	Informações sobre sinônimos no banco de dados
SYS_PRIVS ( DBA_, USER_ )	Privilégios do sistema concedidos a um usuário



TAB_COLUMNS (ALL_, DBA_, USER_ (COLS))	Informações sobre as colunas de tabelas e visões no banco de dados
TAB_COMMENTS ( ALL_, DBA_, USER_ )	Comentários para tabelas e visões no banco de dados
TAB_PRIVS ( ALL_, DBA_, USER_ )	Informações sobre concessões de privilégios de objetos
TAB_PRIVS_MADE ( ALL_, USER_ )	Informações sobre os privilégios de objetos concedidos
TAB_PRIVS_RECD ( ALL_ , USER_ )	Informações sobre privilégios de objetos recebidos
TABLES ( ALL_ , DBA_ , USER_ (TABS))	Informações sobre tabelas no banco de dados
TABLESPACES ( DBA_ , USER_ )	Informações sobre tablespace no banco de dados
TRIGGER_COLS ( ALL_ , DBA_ , USER_ )	Informações sobre as colunas que os gatilhos do banco de dados utilizam
TRIGGERS ( ALL_ , DBA_ , USER_ )	Informações sobre gatilhos no banco de dados
TS_QUOTAS ( DBA_ , USER_ )	Informações sobre cotas de tablespace do usuário
USERS ( ALL_, DBA_, USER_ )	Informações sobre usuários no banco de dados
VIEWS ( ALL_ , DBA_ , USER_ )	Informações sobre visões no banco de dados
WAITERS ( DBA_ )	Informações sobre sessões aguardando em função de um bloqueio mantido pôr outra sessão
AUDIT_ACTIONS	Mapeamento de números de ações de trilha de auditoria para descrição
CHAINED_ROWS	Informações de saída sobre linhas encadeadas a partir do comando ANALYSE. Criado pelo roteiro administrativo UTLCHAIN.SQL
COLUMN_PRIVILEGES	Informações sobre concessões de colunas
DBMS_ALERT_INFO	Informações sobre alertas registrados, criados pelo pacote de utilitários DBMS_ALERT
DBMS_LOCK_ALLOCATED	Informações sobre bloqueios definidos pelo usuários, criados pelo pacote de utilitários DBMS_LOCK
DEPTREE	Informações sobre dependências de objeto. Criado pelo roteiro administrativo UTLDTREE.SQL
DICT_COLUMNS	Informações sobre colunas do dicionário de dados
DICTIONARY (DICT)	Informações sobre tabelas e visões do dicionário de dados
EXCEPTIONS	Informações de saída para exceções de restrições de integridade. Criado pelo roteiro administrativo UTLXPT.SQL
GLOBAL_NAME	Informações sobre o nome global do banco de dados
IDEPTREE	Informações sobre dependências de objeto. Criado pelo roteiro administrativo UTLDTREE.SQL

INDEX_STATS	Informações de estatísticas sobre índices gerados a partir do comando ANALYZE INDEX ... VALIDATE INDEX
LOADER_COL_INFO	Informações para o SQL*Loader relativas a colunas
LOADER_CONSTRAINT_INFO	Informações para o SQL*Loader relativas as restrições de integridade
LOADER_INDCOL_INFO	Informações para o SQL*Loader relativas a colunas de índice
LOADER_IND_INFO	Informações para o SQL*Loader relativas a índices
LOADER_PARAM_INFO	Informações para o SQL*Loader relativas a parâmetros
LOADER_TAB_INFO	Informações para o SQL*Loader relativas a tabelas
LOADER_TRIGGER_INFO	Informações para o SQL*Loader relativas a gatilhos
NLS_DATABASE_PARAMETERS	Informações sobre definições do NLS do banco de dados; útil somente se o servidor usar Nacional Language Support ( NLS – Suporte à Língua Nacional )
NLS_INSTANCE_PARAMETERS	Informações sobre definições do NLS do banco de dados; útil somente se o servidor usar National Language Support (NLS)
NLS_SESSION_PARAMETERS	Informações sobre definições de NLS da sessão ; ; útil somente se o servidor usar National Language Support (NLS)
PLAN_TABLE	Informações de saída sobre planos de execução do otimizador a partir do comando EXPLAIN PLAN. Criado pelo roteiro administrativo UTLXPLAN.SQL
PUBLIC_DEPENDENCY	Informações sobre dependências de objeto
RESOURCE_COST	Informações sobre custos de recursos do sistema
ROLE_ROLE_PRIVS	Informações sobre personagens concedidos a outros personagens
ROLE_SYS_PRIVS	Informações sobre privilégios do sistema concedidos a personagens
ROLE_TAB_PRIVS	Informações sobre privilégios de objeto concedidos a personagens
SESSION_PRIVS	Informações sobre privilégios disponíveis para uma sessão
SESSION_ROLES	Informações sobre personagens disponíveis para uma sessão
STMT_AUDIT_OPTION_MAP	Mapeamento de números de ações de trilha de auditoria para descrições

SYSTEM_PRIVILEGE_MAP	Mapeamento de número de privilégios do sistema para descrições
TABLE_PRIVILEGES	Informações sobre concessões de privilégios de objetos
TABLE_PRIVILEGE_MAP	Mapeamento de números de privilégios de objetos para descrições
V\$ACCESS	Informações sobre objetos atualmente em uso
V\$ARCHIVE	Informações sobre o log de transações arquivado do banco de dados
V\$BACKUP	Informações sobre status de cópia de todos os tablespaces ativos no banco de dados
V\$BGPROCESS	Informações sobre todos os processos de segundo plano do servidor de banco de dados
V\$CIRCUIT	Informações sobre todos os circuitos ( conexões do usuário ) em uma configuração de servidor multilinear
V\$DATABASE	Informações sobre um banco de dados a partir do arquivo de controle do banco de dados
V\$DATAFILE	Informações sobre os arquivos de dados do banco de dados
V\$DBFILE	Informações sobre os arquivos de dados do banco de dados
V\$DB_OBJECT_CACHE	Informações sobre objetos no cache de objetos do servidor de banco de dados, incluindo tabelas, visões, índices e procedimentos
V\$DISPATCHER	Informações sobre processos de segundo plano do despachador do servidor atualmente em andamento em um servidor de banco de dados multilinear
V\$ENABLEDPRIVS	Informações sobre privilégios permitidos
V\$FILESTAT	Informações estatísticas de E/S sobre arquivos do banco de dados
V\$FIXED_TABLE	Informações sobre todas as tabelas fixas no banco de dados
V\$INSTANCE	Informações sobre o estado atual do servidor de banco de dados ( INSTÂNCIA )
V\$LATCH	Informações sobre os bloqueios internos ( serializadores ) no servidor de banco de dados
V\$LATCHHOLDER	Informações sobre sessões que mantêm atualmente bloqueios internos ( serializadores )
V\$LIBRARYCACHE	Informações estatísticas sobre o gerenciamento de caches de bibliotecas

V\$LICENSE	Informações sobre limites de licença de software da Oracle
V\$LOADCTSTAT	Informações sobre estatísticas do SQL*Loader compiladas durante uma carga do caminho direto
V\$LOCK	Informações sobre bloqueios de DML no servidor de banco de dados
V\$LOG	Informações sobre o log de transações de um banco de dados
V\$LOGFILE	Informações sobre arquivos de log de transações de um banco de dados
V\$LOGHIST	Informações sobre histórico de sequência de log de transações de um banco de dados
V\$LOG_HISTORY	Informações Sobre o log de transações de um banco de dados
V\$MTS	Informações de ajuste para configuração de servidor multilinear
V\$NLS_PARAMETERS	Informações sobre valores atuais de parâmetros do Suporte de Língua Nacional
V\$OPEN_CURSOR	Informações sobre cursores abertos de cada sessão do banco de dados
V\$PARAMETER	Informações sobre cada parâmetro de inicialização do servidor de banco de dados
V\$PROCESS	Informações sobre processos atualmente ativos
V\$QUEUE	Informações sobre filas do servidor multilinear
V\$RECOVER_LOG	Informações sobre os grupos de log de transações arquivados, necessários para fazer a recuperação do banco de dados
V\$REQDIST	Informações estatísticas sobre tempos de solicitação
V\$RESOURCE	Informações sobre recursos do sistema
V\$ROLLNAME	Informações sobre todos os segmentos reconstitutivos ativos
V\$ROLLSTAT	Informações estatísticas sobre todos os segmentos reconstitutivos ativos
V\$ROWCACHE	Informações estatísticas sobre atividade do dicionário de dados
V\$SECONDARY	Informações sobre banco de dados secundários montados; útil somente com Trusted Oracle7
V\$SESSION	Informações sobre sessões do banco de dados

V\$SESSION_EVENTS	Informações estatísticas de espera para cada sessão e evento
V\$SESSION_WAIT	Informações sobre recursos que as sessões estejam esperando
V\$SESSTAT	Informações estatísticas sobre sessões do banco de dados
V\$SESS_IO	Informações sobre o uso de E/S de cada sessão
V\$SGA	Informações sobre a área de memória SGA do servidor de banco de dados
V\$SGASTAT	Informações estatísticas sobre a área de memória SGA do servidor de banco de dados
V\$SHARED_SERVER	Informações sobre os servidores de primeiro plano compartilhados de um servidor de banco de dados multi-linear
V\$SQLAREA	Informações sobre cursores compartilhados
V\$SQLTEXT	Informações sobre instruções correspondentes a cursores compartilhados
V\$STATNAME	Descrições para códigos de estatísticas de sessão mostrados em V\$SESSTAT
V\$SYSSTAT	Informações sobre rótulos do sistema; útil somente com o Trusted Oracle7
V\$SYSTEM_EVENTS	Informações do sistema sobre todas as estatísticas de eventos por sessão
V\$THREAD	Informações sobre linhas do log de transações do banco de dados
V\$TIMER	A hora atual do sistema em centésimos de segundo
V\$TRANSACTION	Informações sobre as transações atuais do banco de dados
V\$TYPE_SIZE	Informações sobre componentes de dados de baixo nível para ajudar na projeção de estimativas de uso de espaço
V\$VERSION	Informações sobre versões de bibliotecas centrais de software do servidor Oracle7
V\$WAITSTAT	Informações estatísticas sobre contenção de bloqueio de dados entre transações

## Comandos D.M.L.

São comandos utilizados para selecionar, incluir, alterar, excluir dados nas tabelas e que precisam de um COMMIT ou ROLLBACK para terminar a transação.

### Inserindo Dados Na Tabela

O comando utilizado para adicionar linhas a uma tabela é o INSERT.

Sintaxe:

```
INSERT INTO tabela [coluna, coluna, ...]  
VALUES (valor1, valor2, ...);
```

Exemplo:

```
SQL> desc dept  
Name                Null?  Type  
-----  
DEPTNO              NOT NULL NUMBER(2)  
DNAME                VARCHAR2(14)  
LOC                 VARCHAR2(13)  
  
SQL> insert into dept (deptno, dname, loc)  
2  values (50, 'MARKETING', 'FORTALEZA');  
1 row created.
```

Inserindo dados a partir de outra tabela.

Exemplo:

```
SQL> insert into dept (deptno, dname, loc)  
2  select deptno, dname, loc from scott.dept  
3  where loc='SÃO PAULO';  
10 rows created.
```

## Alterando Dados Na Tabela

O comando utilizado para alterar os dados de uma tabela é o UPDATE.

Sintaxe:

```
UPDATE tabela [alias]
SET      coluna [,coluna, ...] = { expressão, subquery}
[WHERE condição];
```

Exemplo:

```
SQL> select * from dept;
DEPTNO DNAME      LOC
-----
 10 ACCOUNTING   NEW YORK
 20 RESEARCH     DALLAS
 30 SALES         CHICAGO
 40 OPERATIONS    BOSTON
 50 MARKETING     São Paulo
```

```
SQL> update dept
2  set loc = 'Fortaleza'
3* where deptno = 50;
```

1 row updated.

```
SQL> select * from dept;
```

```
DEPTNO DNAME      LOC
-----
 10 ACCOUNTING   NEW YORK
 20 RESEARCH     DALLAS
 30 SALES         CHICAGO
 40 OPERATIONS    BOSTON
 50 MARKETING     Fortaleza
```

## Excluindo Linhas Da Tabela

O comando para excluir linhas de uma tabela é o DELETE.

Sintaxe:

```
DELETE FROM tabela  
[WHERE condição];
```

Exemplo:

```
SQL> select * from dept;  
  
DEPTNO DNAME      LOC  
-----  
  10 ACCOUNTING  NEW YORK  
  20 RESEARCH    DALLAS  
  30 SALES        CHICAGO  
  40 OPERATIONS  BOSTON  
  50 MARKETING   Fortaleza  
  
SQL> delete from dept  
2  where deptno = 50;  
  
1 row deleted.  
  
SQL> select * from dept;  
  
DEPTNO DNAME      LOC  
-----  
  10 ACCOUNTING  NEW YORK  
  20 RESEARCH    DALLAS  
  30 SALES        CHICAGO  
  40 OPERATIONS  BOSTON
```



**Observações importantes:**

1. Na alteração e na exclusão não se pode esquecer a cláusula WHERE sob pena da tabela ser totalmente alterada ou excluída.
2. Para validar os comandos D.M.L. (inclusão, alteração e exclusão) encerre a transação com um COMMIT ou ROLLBACK.

# **Controle de transações**

Existem dois tipos de transações: as DML originadas pelos comandos de manipulação de dados e as DDL geradas pelos comandos de manipulação de estrutura das tabelas.

As DDL são formadas por um único comando e tem um COMMIT implícito.

As DML são formadas por um ou mais comandos e não têm COMMIT.

O COMMIT confirma ao ORACLE o fim da transação.

O ROLLBACK é utilizado para desfazer a transação.

Dentro de uma transação podemos marcar pontos de salvaguarda.

## **Controlando As Transações Com Comandos Sql**

### **COMMIT**

---

Sintaxe:

COMMIT;

1. Atualiza o banco permanentemente;
2. Limpa os SAVEPOINTS da transação;
3. Finaliza a transação;
4. Libera os LOCK's das tabelas;
5. Antes de um comando DDL;
6. Depois de um ou mais comando DML;
7. Na desconexão do banco de dados.

### **SAVEPOINT**

---

Sintaxe:

SAVEPOINT nome\_savepoint;

1. É utilizado para dividir a transação em pequenos blocos.
2. O número máximo de savepoints por processo de usuário é cinco, mas pode ser alterado.
3. Se for criado um savepoint com o mesmo nome de um anterior, o anterior será apagado.

---

## ROLLBACK

---

É utilizado para desfazer uma transação.

Sintaxe:

ROLLBACK [WORK] [ to SAVEPOINT nome\_savepoint];

1. A palavra work é opcional;
2. São desfeitas todas as modificações da transação.
3. Limpa todos os savepoints da transação;
4. Libera os LOCK's de transação.

[illegible]

## **DESENVOLVIMENTO DE APLICAÇÕES com PROCEDURAL OPTIONS**

O **PL/SQL** é uma linguagem criada para interagir com Bases de Dados e possuir comandos típicos de linguagens de terceira geração, tais como IF .....THEN ....ELSE.

Com esta linguagem pode-se escrever triggers para o Forms com grande flexibilidade. PL/SQL é uma extensão da linguagem SQL e, portanto, permite utilizar os comandos de manipulação de dados (SELECT, INSERT, UPDATE e DELETE), bem como operações com cursores (COMMIT, ROLLBACK e SAVEPOINT).

A principal vantagem do PL/SQL é permitir um aumento de desempenho no acesso à Base de Dados. Isto ocorre porque os comandos da linguagem SQL devem ser processados um por vez pelo RDBMS. Com a linguagem PL/SQL, um bloco inteiro de comandos pode ser enviado ao RDBMS para processamento. Em ambientes de rede, com vários acessos à Base de Dados, a diferença de desempenho torna-se significativa. Além disso, com o uso do PL/SQL há uma diminuição na necessidade de consultas à Base de Dados devido à capacidade dos comandos da linguagem.

Outra vantagem do PL/SQL é permitir que um trigger do Forms seja totalmente composto por um bloco em PL/SQL, sem a necessidade de vários passos, macros ou user exits.

As aplicações em PL/SQL são portáteis para qualquer computador e sistema operacional que contenha o Oracle Server.

## **Conceitos Básicos**

As aplicações em PL/SQL são normalmente feitas para serem ativadas a partir de triggers do Forms. Entretanto, elas também podem ser executadas a partir do SQL\*Plus. Para isto, constrói-se a aplicação diretamente com o editor do SQL\*Plus ou carrega-se o arquivo texto. Após digitado ou carregado o bloco, teclando-se um ponto (.) armazena-se o arquivo no buffer do SQL. Se ao invés do ponto for teclada uma barra (/), além de armazenar o arquivo no buffer do SQL, ele é executado.

Se o buffer do SQL contém um arquivo, para executá-lo o procedimento é igual ao empregado para um arquivo de comandos SQL: basta teclar RUN e [ENTER], ou simplesmente / e [ENTER].

Uma linha de comando em PL/SQL pode ser escrita em letras maiúsculas ou minúsculas. É costume escrever os comandos de PL/SQL em maiúsculas e nomes de tabelas, variáveis e outros objetos em minúsculas. Toda linha em PL/SQL deve terminar com um ponto-e-vírgula quando encerra um comando.

### **VARIÁVEIS**

Assim como em linguagens de terceira geração, o PL/SQL permite a declaração de variáveis e constantes. Variáveis e constantes somente podem ser usadas em um bloco PL/SQL após terem sido declaradas.

Os tipos de variáveis e constantes são os mesmos do Oracle Server (NUMBER, CHAR E DATE), acrescidos de um tipo do PL/SQL: BOOLEAN.

A declaração de um variável é feita da seguinte maneira:

```
nome_var TIPO [(tamanho)];
```

onde:

- **nome\_var**

É o nome da variável; pode ser usado qualquer nome que obedeça as regras de construção de nomes no ORACLE.

- **TIPO**

É o tipo da variável, podendo assumir os valores:

- CHAR
- NUMBER
- DATE
- BOOLEAN

- **TAMANHO**

É o tamanho da variável, e aplicável apenas para as variáveis do tipo CHAR e NUMBER.

A seguir são dados exemplos de declaração de variáveis em PL/SQL.

```
funcionário    CHAR (15);  
admissão      DATE;  
codigo_func    NUMBER (4);  
decisão        BOOLEAN;
```

Existem duas maneiras de se atribuir um valor a uma variável em PL/SQL.

1. Empregar o operador de atribuição := (dois pontos igual), semelhante ao do PASCAL. Neste caso, no lado direito deste comando pode haver qualquer expressão composta de constantes, variáveis, operadores algébricos e funções do PL/SQL.
2. Fazendo recuperação de dados diretamente na variável, através dos comandos SELECT ou FETCH.

A seguir são mostrados exemplos de atribuições de valores a variáveis.

```
codigo_func := 104;
funcionário := ALBERTO ;
SELECT cod_func INTO codigo_func FROM FUNCIONÁRIO
  WHERE nome_func = ROSANGELA;
preço_venda := preço_compra*1.5;
```

- Podem ser aplicadas às variáveis, bem como às constantes, as funções do SQL para manipulação de cadeias de caracteres, números, datas, grupos e conversões.

## CONSTANTES

A declaração de uma constante é similar à de uma variável, exceto pela obrigatoriedade da palavra **CONSTANT** e da atribuição imediata de um valor a ela. Deste ponto em diante, não é permitido mudar o valor de uma constante.

Exemplos de declarações de constantes são dados a seguir.

```
fator_lucro CONSTANT NUMBER (3,2) := 1.50;
```

Com esta declaração, o cálculo do preço de venda do produto, feito logo acima, tomaria a seguinte forma:

```
preço_venda := preço_compra*fator_lucro;
```

## OUTROS TIPOS DE VARIÁVEIS

Existe uma outra forma de declaração bastante útil quando se deseja associar uma variável a uma coluna de uma tabela e não se conhece com exatidão a definição desta coluna. Nestas ocasiões, utiliza-se o atributo **%TYPE**.

Esta declaração é da seguinte forma:

```
var tab.col%TYPE;
```

Assim, a variável **var** terá o mesmo tipo da coluna **col** da tabela **tab**, sem que o operador precise saber que tipo é este.

Outra vantagem desta declaração é que se a coluna **col** for mudada (por exemplo em seu tamanho), a variável **var** mudará conjuntamente.

Outra forma de declaração existente é **%ROWTYPE**. Este tipo é usado quando se declara um cursor para uma busca. Uma declaração típica seria:



```
DECLARE
  CURSOR cur_func IS SELECT nome_func, cod_func, salário
    FROM funcionário;
```

Após esta declaração, `cur_func` é definido como um cursor e pode-se declarar uma variável para receber os dados das buscas deste cursor da seguinte maneira:

```
dados_func cur_func%ROWTYPE;
```

Com esta declaração, a variável `dados_func` torna-se do tipo registro, podendo receber os dados recuperados pela busca do cursor `cur_func`. Assim, após o comando:

```
FETCH cur_func INTO dados_func;
```

têm-se três variáveis, a saber:

- `dados_func.nome_func` - contém o nome do funcionário;
- `dados_func.cod_func` - contém o código do funcionário;
- `dados_func.salário` - contém o salário do funcionário.

Para se definir dados novos na variável `dados_func` é necessário utilizar os comandos `SELECT ... INTO` ou `FETCH ... INTO`. Entretanto, é possível modificar apenas um dos campos de uma variável tipo registro com um comando de atribuição simples:

```
dados_func.salário := <expressão>;
```

Existe também a possibilidade de uma declaração implícita, que ocorre quando se tem um laço de cursor `FOR` (`CURSOR FOR LOOP`). Por exemplo, se foi declarado o cursor `cur_func` como mostrado anteriormente, no comando seguinte.

```
FOR dados_func IN cur_func LOOP
  folha_pag := folha_pag + dados_func.salário;
END LOOP;
```

a variável `dados_func` é implicitamente declarada do tipo `cur_func%ROWTYPE`.

## CURSORES

Uma área de contexto é uma área de trabalho aberta pelo PL/SQL para processar comandos SQL e para armazenar informações de processamento. Um cursor é uma entidade do PL/SQL que permite nomear uma área de contexto, acessar as informações armazenadas e, em alguns casos, controlar seu processamento. Em PL/SQL existem dois tipos de cursores: os explícitos e os implícitos.

Cursores explícitos são aqueles declarados explicitamente para obtenção de buscas que resultem em vários registros. Cursores implícitos são definidos para todos os outros comandos SQL.

Quatro são os comandos de PL/SQL usados para manipular um cursor explícito: DECLARE, OPEN, FETCH E CLOSE. As definições destes comandos são dadas a seguir.

- **DECLARE**

É o comando empregado para declarar um cursor. Para isto coloca-se um nome e o comando SELECT correspondente.

Por exemplo:

```
DECLARE
  CURSOR busca1 IS SELECT nome_dep, local
                    FROM depto
                    WHERE cod_dep = 20;
```

- **OPEN**

Com este comando executa-se a busca declarada com o comando anterior e identifica-se o conjunto ativo (todas as linhas e colunas que atendem ao critério de busca estabelecido). Por exemplo:

```
OPEN busca1;
```

- **FETCH**

Este comando recupera cada linha do conjunto ativo, uma por vez. Cada execução deste comando faz o cursor avançar para a próxima linha no conjunto ativo. Por exemplo:

```
FETCH busca1 INTO codigo_depto, nome_depto;
```

- **CLOSE**

Este comando desativa o cursor indicado, tornando indefinido o conjunto ativo. Por exemplo:

```
CLOSE busca1;
```

Existem quatro atributos que podem ser usados para acessar a área de contexto de cursores ativos: %FOUND, %NOTFOUND, %ROWCOUNT e %ISOPEN. Esses atributos são usados para se obter informações a respeito da execução de buscas. Para usar um atributo basta agregá-lo ao nome do cursor. Os significados de cada atributo são:

- **%FOUND**  
Este atributo retorna TRUE se o último comando FETCH foi bem sucedido, ou seja, recuperou uma linha, ou FALSE se aquele comando falhou, ou seja, não recuperou nenhuma linha.
- **%NOTFOUND**  
Este atributo é o oposto lógico de %FOUND.
- **%ROWCOUNT**  
Este atributo retorna o número de linhas recuperadas pelo conjunto ativo até o momento.
- **%ISOPEN**  
Este atributo retorna TRUE se um cursor explícito está aberto e FALSE caso contrário.

Se o comando SQL não é um SELECT, informações importantes ficam armazenadas em áreas de contexto do Oracle, a respeito da execução dos comandos INSERT, UPDATE, DELETE e SELECT que recupere apenas uma linha. Para acessar estas áreas, pode-se referenciar a área de contexto implícita mais recente através do cursor SQL%. Este é o chamado cursor implícito. Também para cursores implícitos existem quatro atributos que podem ser usados para acessar a área de contexto: %FOUND, %NOT\_FOUND, %ROWCOUNT e %ISOPEN. Estes atributos são usados para se obter informações a respeito da execução de buscas.

- **SQL%FOUND**  
Este atributo retorna TRUE se o último comando INSERT, UPDATE ou DELETE foi bem sucedido, ou seja, afetou pelo menos uma linha ou um SELECT de registro único retornou uma linha, ou FALSE se aquele comando falhou.
- **SQL%NOTFOUND**  
Este atributo é o oposto lógico de %FOUND.
- **SQL%ROWCOUNT**  
Este atributo retorna o número de linhas afetadas pelo último comando INSERT, UPDATE, DELETE ou SELECT de registro único.
- **SQL%ISOPEN**  
Este atributo sempre retorna FALSE porque o Oracle automaticamente fecha um cursor implícito após executar o comando SQL associado a ele.

## **Blocos em PL/SQL**

Um bloco em PL/SQL é dividido basicamente em três partes: uma parte de declarações, uma parte com comandos executáveis e uma parte com manipuladores de exceções. A ordem destas partes deve ser a apresentada acima.

Em alguns casos é preciso dar nome a um bloco, para posterior referência a ele. Para fazer isto, antes da parte de declarações coloca-se:

```
<<nome_bloco>>
```

Neste caso, no final do bloco, após a palavra END deve-se colocar o nome do bloco, da seguinte maneira:

```
END nome_bloco;
```

Um bloco pode conter outros blocos, isto é, pode haver blocos aninhados, chamados sub-blocos. Estes sub-blocos podem estar na parte executável ou na de manipulação de exceções, mas não na de declarações.

- Identificadores são nomes de objetos em PL/SQL, ou sejam, constantes, variáveis, registros, cursores e exceções.
- Um identificador declarado em um bloco somente é válido dentro deste bloco.
- Se em um sub-bloco é declarado um identificador com o mesmo nome, tem prioridade o identificador de nível mais interno.
- Identificadores declarados em um bloco são locais a ele.
- Se este bloco é um sub-bloco, os identificadores declarados no bloco mais externo são chamados globais.
- A maneira de acessar um identificador global, quando existe um identificador local de mesmo nome, é utilizar o nome, ou rótulo, do bloco, da seguinte forma:

```
bloco.identificador.
```

### **DECLARE**

Declaração de variáveis

**BLOCO PADRÃO PL/SQL**

### **BEGIN**

Processamento dos comandos SQL e PL/SQL

### **EXCEPTION**

Tratamento das exceções definidas

**END;**

## **Manipuladores de Exceções**

Chama-se exceção quando o processamento normal de um bloco em PL/SQL é suspenso e a execução passa para os comandos declarados para este fim (manipular a exceção). Após a execução do manipulador de exceção o controle passa de volta ao ponto de onde foi ativada a exceção.

Uma exceção é ativada quando ocorre um erro que torna impossível ou indesejável continuar com o processamento, ou então quando o usuário deseja explicitamente que, ocorrendo certas condições, o processamento do bloco seja interrompido e sejam executados certos comandos.

A definição dos manipuladores de exceção é feita no final da parte executável do bloco. A sintaxe é a seguinte:

```
EXCEPTION
  WHEN nome_exce OR nome_exce ... THEN
    comandos;
  WHEN ...
END;
```

Como se nota, uma exceção deve ser encerrada pela palavra END. Os nomes de exceção e seus significados são os seguintes:

### **DUP\_VAL\_ON\_INDEX**

Esta exceção é ativada quando um comando INSERT ou UPDATE tenta criar duas linhas com o mesmo valor em colunas restritas pelo índice UNIQUE. SQLCODE retorna -1, o que equivale ao código de erro Oracle ORA-00001.

### **INVALID\_CURSOR**

Esta exceção é ativada quando uma chamada em PL/SQL especifica um cursor inválido. SQLCODE retorna -1001, equivale ao código de erro Oracle ORA-01001.

### **INVALID\_NUMBER**

Esta exceção é ativada quando é tentada uma conversão de uma cadeia de caracteres para um número e a cadeia contém caracteres não válidos. SQLCODE -1722, o que equivale ao código de erro Oracle ORA-01722.

### **LOGIN\_DENIED**

Esta exceção é ativada quando é usada uma combinação inválida de Usuário/Senha para entrada no Oracle. SQLCODE retorna -1017, o que equivale ao código de erro Oracle ORA-01017.

---

**NO\_DATA\_FOUND**

---

Esta exceção é ativada quando um comando SELECT não retorna colunas. SQLCODE retorna +100, o que equivale ao código de erro Oracle ORA-01403.

---

**NOT\_LOGGED\_ON**

---

Esta exceção é ativada quando o PL/SQL efetua uma chamada ao Oracle sem estar logado nele. SQLCODE retorna -1012, o que equivale ao código de erro Oracle ORA-01012.

---

**PROGRAM\_ERROR**

---

Esta exceção é ativada quando ocorre um problema interno no PL/SQL. SQLCODE retorna -6501, o que equivale ao código de erro Oracle ORA-06501.

---

**STORAGE\_ERROR**

---

Esta exceção é ativada quando o PL/SQL excede a memória ou quando a memória está corrompida. SQLCODE retorna -6500, o que equivale ao código de erro Oracle ORA-06500.

---

**TIMEOUT\_ON\_RESOURCE**

---

Esta exceção é ativada quando ocorre um “timeout” enquanto o Oracle XXXX.

---

**TOO\_MANY\_ROWS**

---

Esta exceção é ativada quando um comando SELECT retorna mais de uma linha. SQLCODE retorna -1427, o que equivale ao código de erro Oracle ORA-01427.

---

**VALUE\_ERROR**

---

Esta exceção é ativada quando ocorrem erros aritméticos, numéricos, com caracteres, de conversão ou de restrições. Também é ativada quando um comando INSERT ou UPDATE interage com a Base de Dados com uma string e ela é truncada, ou se uma string é truncada quando atribuída a uma variável PL/SQL. Se uma variável hospedeira é truncada, entretanto, não é ativada esta exceção. SQLCODE retorna ORA-06502.

---

**ZERO\_DIVIDE**

---

Esta exceção é ativada quando se tenta dividir um número por zero. SQLCODE retorna -1476, o que equivale ao código de erro Oracle ORA-01476.

---

**OTHERS**

---

Usa-se esta exceção para levar em conta todas as outras exceções não explicitamente mencionadas na parte dos manipuladores de exceções.

Além destas exceções outras podem ser definidas pelo próprio usuário. Sua declaração é feita com o nome seguido da palavra EXCEPTION, da seguinte forma:

```
sem_registro      EXCEPTION;
```

Para ativar este tipo de exceção é preciso fazê-lo explicitamente com o comando RAISE, da seguinte maneira:

```
RAISE sem_registro;
```

## **Comandos do PL/SQL**

A seguir são listados e descritos os comandos do PL/SQL, ordenados alfabeticamente.

### **BEGIN**

Este é o comando que sinaliza o início da parte executável de um bloco. Deve haver pelo menos um comando executável após o BEGIN.

### **CLOSE**

Este comando fecha um cursor, ou seja, libera os recursos reservados por um cursor aberto. O uso é da seguinte maneira:

```
CLOSE cursor ;
```

### **COMMIT**

Este comando torna permanentes todas as alterações feitas na Base de Dados desde a última transação deste tipo. Com este comando, também, as alterações na Base de Dados tornam-se visíveis para outros usuários. Opcionalmente pode-se colocar a palavra WORK após COMMIT. Este comando também libera travas em linhas e tabelas e elimina os declarados desde a última transação como esta. Se é solicitado COMMIT enquanto está aberto um cursor como SELECT FOR UPDATE, qualquer FETCH subsequente resultará em erro.

### **DECLARE**

Este é o comando que sinaliza o início da parte de declarações. Nesta parte são declaradas as variáveis, constantes, cursores e exceções. Na declaração de cursores, se houver parâmetros, é preciso declará-los e a seus tipos. A sintaxe do comando é:

```
DECLARE nome_var tipo;  
DECLARE nome_var CONSTANT tipo := expressão;  
DECLARE nome_exce EXCEPTION;  
DECLARE CURSOR nome_cur [(par1 tipo, ...)] IS  
comando select [FOR UPDATE OF coluna]
```

Nesta última forma, podem ser passados parâmetros ao comando SELECT, normalmente para serem usados na cláusula WHERE. Também nesta forma, a cláusula FOR UPDATE OF coluna é usada em conjunto com a cláusula WHERE CURRENT OF nome\_var dos comandos UPDATE ou DELETE, permitindo atualizar ou eliminar a última linha recuperada.

### **DELETE**



Com este comando eliminam-se linhas inteiras de uma tabela ou visão. A sintaxe do comando é:

```
DELETE FROM tabela WHERE { condição | CURRENT OF nome_cur };
```

Utilizando a forma WHERE CURRENT OF nome\_cur, elimina-se a última linha recuperada por um comando FETCH.

---

## END

Com este comando encerra-se um bloco PL/SQL, um comando LOOP ou um comando IF.

---

## EXCEPTION

Este comando sinaliza o início da parte de manipuladores de exceções de um bloco PL/SQL.

---

## EXCEPTION\_INIT

Este comando designa um nome para um código de erro Oracle, permitindo referências a exceções internas por nome ao invés de usar OTHERS no manipulador de exceções. A sintaxe do comando é:

```
PRAGMA EXCEPTION_INIT ( nome_exc; num_cod_err )
```

onde

- **PRAGMA**  
Significa que o comando não é processado quando o bloco PL/SQL é executado.
- **nome\_exc**  
É o nome de uma exceção previamente definida.
- **num\_cod\_err**  
É qualquer código de erro válido Oracle.

---

## EXIT

É o comando usado para sair de um laço. Pode ser incondicional ou condicional, na forma EXIT WHEN condição.

---

## FETCH

---

Com este comando é recuperada a próxima linha de dados do conjunto ativo de um cursor. Estes dados são armazenados em variáveis. A sintaxe do comando é:

```
FETCH nome_cur INTO var1 [, var2, ...] [nome_reg]
```

onde

- **nome\_cur**  
É o nome do cursor que faz a recuperação dos dados.
- **var1, var2**  
São os nomes das variáveis a receber os dados recuperados.
- **nome\_reg**  
É uma variável do tipo registro.

---

## GOTO

---

Este comando transfere o controle de fluxo do programa para o comando cujo rótulo é colocado após este comando. A sintaxe do comando é:

```
GOTO rótulo;
```

---

## IF

---

Este comando executa uma sequência de comandos condicionalmente; a sequência a ser executada depende da condição especificada. A sintaxe do comando é a seguinte:

```
IF condição THEN comandos  
ELSIF condição THEN comandos  
    ELSE comandos  
END IF;
```

---

## INSERT

---

Com este comando inserem-se linhas de dados na tabela ou visão especificada. A sintaxe do comando é igual a do comando INSERT do SQL:

```
INSERT INTO tabela (coluna1, ...) VALUES  
(expressões);
```

---

## LOCK TABLE

---

Este comando permite travar uma ou mais tabelas em um modo determinado. A sintaxe do comando é:

```
LOCK TABLE tabela IN modo MODE [NOWAIT];
```

onde

- **tabela**  
Pode ser o nome de uma ou mais tabelas ou visões.
- **modo**  
Este parâmetro especifica o modo de travamento da tabela: ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE, SHARE, SHARE ROW EXCLUSIVE ou EXCLUSIVE.
- **NOWAIT**  
Este parâmetro especifica que se uma tabela não pode ser travada no instante da solicitação, então o controle deve retornar ao usuário ao invés de esperar até a efetivação da trava.

## LOOP

Com este comando consegue-se executar um conjunto de comandos zero ou mais vezes. A construção do comando deve conter todos os comandos a serem executados. Existem quatro tipos de laços:

- laços básicos
- laços WHILE
- laços FOR numéricos
- laços FOR de cursores

Os laços básicos são do tipo:

```
LOOP  
comandos  
END LOOP;
```

O laço neste caso é infinito e é necessário que dentro dos comandos haja uma forma de encerrar o laço.

Os laços WHILE são do tipo:

```
WHILE condição LOOP  
comandos  
END LOOP;
```

Os laços FOR numéricos são do tipo:

```
FOR param [ REVERSE ] IN val1..val2 LOOP  
comandos  
END LOOP;
```

A palavra REVERSE faz o laço começar no valor val2 e decrementar param até val1.

Os laços FOR de cursores são do tipo:

```
FOR var_reg IN nome_cur LOOP  
comandos  
END LOOP;
```

Neste caso, enquanto for possível atribuir linhas recuperadas pelo cursor nome\_cur à variável var\_reg o laço é realizado.

## NULL

Este comando nada faz e somente tem a ação de passar o controle ao próximo comando. A única razão de existir este comando é para melhoria da legibilidade.

## OPEN

Este comando efetua a busca associada a um cursor explícito declarado. As linhas que satisfazem a condição de busca são identificadas e recebem o nome de conjunto ativo. A sintaxe do comando é:

```
OPEN nome_cur [ parâmetros ];
```

onde os parâmetros passados devem ter sido referenciados no comando DECLARE.

## RAISE

Este comando suspende a execução normal do bloco PL/SQL e transfere o controle para o manipulador de exceções que trata daquela exceção específica, caso tenha sido declarada. Após o tratamento desta exceção o controle passa ao bloco que contém o bloco onde foi ativada a exceção. A sintaxe deste comando é:

```
RAISE nome_exceção;
```

## ROLLBACK

Este comando desconsidera todas ou algumas das alterações feitas à Base de Dados desde que as últimas alterações foram tornadas permanentes. Existem duas formas de se usar este comando:

```
ROLLBACK
```

Nesta forma o comando desconsidera todas as alterações feitas à Base de Dados desde a última validação das modificações.

```
ROLLBACK TO
```

Nesta forma o comando desconsidera apenas as alterações feitas à Savepoint Base de Dados desde o SAVEPOINT mencionado.

## SAVEPOINT

Com este comando coloca-se uma “marca” no ponto atual do processamento de uma transação e dá-se-lhe um nome. Este comando normalmente é usado em conjunto com o ROLLBACK TO.

## SELECT...INTO

Este comando recupera dados de uma tabela ou visão e os armazena em uma variável. A sintaxe do comando é:

```
SELECT expressão INTO variável FROM tabela resto_comando;
```

---

## SET TRANSACTION

---

Este comando permite estabelecer uma transação como sendo apenas de leitura (READ-ONLY). As buscas subsequentes somente enxergam mudanças que tenham sido efetivadas antes do início da transação. A sintaxe do comando é:

```
SET TRANSACTION READ ONLY;
```

---

## UPDATE

---

Este comando atualiza os valores especificados nas colunas da tabela ou visão. A sintaxe do comando é:

```
UPDATE tabela SET coluna = expressão [WHERE {condição | CURRENT OF nome_cur  
} ];
```

Utilizando a forma WHERE CURRENT OF nome\_cur, atualiza-se a última linha recuperada por um comando FETCH.

# Desenvolvendo STORED PROCEDURES e Funções

## CRIANDO PROCEDURES

Para criarmos uma Procedure, devemos utilizar o comando CREATE PROCEDURE, declarar a lista de argumentos e definir a ação a ser executada pelo bloco PL/SQL.

Sintaxe:

```
CREATE [OR REPLACE] PROCEDURE [schema.]procedure _name  
  [ (argumento [IN | OUT | IN OUT] datatype  
    [, argument o [IN | OUT | IN OUT] datatype] ...)]  
  IS | AS} pl/sql_subprogram_body
```

onde :

- **[schema.] procedure\_name**  
Determina o nome da procedure.
- **argumento**  
Determina o nome da variavel PL/SQL passada como parametro para a procedure.
- **IN | OUT | IN OUT**  
Determina o tipo de argumento.

TIPO DE ARGUMENTO	DESCRIÇÃO
IN (default)	é o argumento (parametro) de entrada
OUT	é o argumento de retorno
IN OUT	é o argumento de entrada e saída

- **datatype**  
Tipo de dado do argumento
- **pl/sql\_subprogram\_body**  
Bloco PL/SQL que determina a ação realizada na procedure com o argumento recebido.

Deve ser escrito da seguinte forma:

```
IS / AS (substituindo o DECLARE)
    definição das variáveis locais
BEGIN
    corpo da procedure
EXCEPTION
    tratamento das exceções
END nome_procedure
Utilizar REPLACE se a procedure já existir.
```



**Criação de Procedures com tipo IN de argumentos**

Procedure para inserir dados em um cadastro:

```
CREATE OR REPLACE PROCEDURE hire_emp
  ( v_empno  IN emp.empno%TYPE,
    v_ename  IN emp.ename%TYPE,
    v_job    IN emp.job%TYPE,
    v_mgr    IN emp.mgr%TYPE,
    v_hiredate IN emp.hiredate%TYPE,
    v_sal     IN emp.sal %TYPE,
    v_comm    IN emp.comm%TYPE,
    v_deptno  IN emp.deptno%TYPE)
IS
BEGIN
  INSERT INTO emp
  VALUES (v_empno, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno);
  COMMIT;
END hire_emp;
```

### Eliminando Parâmetros Desnecessários de Entrada

- Gere as primary key usando SEQUENCE do database;
- Utilize SYSDATE para atualizar datas correntes;
- Armazene valores default apropriadamente;
- Utilize as regras de negocio para implementar valores automaticamente.

```
CREATE OR REPLACE PROCEDURE hire_emp
  ( v_ename IN emp.ename%TYPE,
    v_job    IN emp.job%TYPE,
    v_mgr    IN emp.mgr%TYPE,
    v_sal     IN emp.sal %TYPE)
IS
  v_hiredate emp.hiredate%TYPE;
  v_comm     emp.comm%TYPE;
  v_deptno   emp.deptno%TYPE;
BEGIN
  v_hiredate := sysdate;
  IF v_job = 'SALESMAN' THEN
    v_comm := 0;
  ELSE
    v_comm := NULL;
  END IF
  SELECT deptno INTO v_deptno FROM emp WHERE empno=v_mgr;
  INSERT INTO emp
  VALUES (seq_empno.nextval, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno);
  COMMIT;
END hire_emp;
```

**Criação de Procedures com tipo OUT de argumentos**

```
CREATE OR REPLACE PROCEDURE query_emp
  ( v_empno IN    emp.empno%TYPE,
    v_ename OUT  emp.ename%TYPE,
    v_job      OUT emp.job%TYPE,
    v_sal      OUT emp.sal %TYPE,
    v_comm     OUT emp.comm%TYPE)
IS
BEGIN
  SELECT  ename, job, sal, comm INTO v_ename, v_job, v_sal, v_comm
  FROM emp
  WHERE empno=v_empno;
END query_emp;
```

A procedure atualiza os parâmetros OUT mas não retorna valor.

**Criação de Procedures com tipo IN OUT de argumentos**

```
PROCEDURE calc_bonus (emp_id IN INTEGER, bonus IN OUT REAL)
IS
  hire_date DATE;
  bonus_missing EXCEPTION;
BEGIN
  SELECT sal * 0.10, hiredate INTO bonus, hire_date FROM emp
  WHERE empno = emp_id;
  IF bonus IS NULL THEN
    RAISE bonus_missing;
  END IF;
  IF MONTHS_BETWEEN(SYSDATE, hire_date) > 60 THEN
    bonus := bonus + 500;
  END IF;
  ...
EXCEPTION
  WHEN bonus_missing THEN
    ...
END calc_bonus;
```

A procedure atualiza os parâmetros IN OUT mas não retorna valor.

## CRIANDO FUNÇÕES

Ao contrário das procedures as funções sempre retornam um valor. O comando para criar uma função é CREATE FUNCTION, declarar a lista de argumentos e definir a ação a ser executada pelo bloco PL/SQL.

Sintaxe:

```
CREATE [OR REPLACE] FUNCTION [schema.]function _name  
  [ (argumento [IN ] datatype  
    [, argument o [IN ] datatype] ...)]  
  RETURN CHAR | NUMBER | BOOLEAN  
  IS | AS} pl/sql_subprogram_body
```

onde :

- **[schema.] function\_name**  
Determina o nome da função.
- **argumento**  
Determina o nome da variavel PL/SQL passada como parametro para a função.
- **IN**  
Determina o tipo de argumento, como parametro de entrada,
- **datatype**  
Tipo de dado do argumento
- **RETURN**  
Determina o que vai retornar como resultado da função
- **pl/sql\_subprogram\_body**  
Bloco PL/SQL que determina a ação realizada na função com o argumento recebido.

Deve ser escrito da seguinte forma:

```
IS / AS (substituindo o DECLARE)  
  definição das variáveis locais  
BEGIN  
  corpo da função  
EXCEPTION  
  tratamento das exceções  
END nome_função  
  
Utilizar REPLACE se a função já existir.
```

Exemplo:

```
CREATE OR REPLACE FUNCTION get_sal  
  ( v_empno IN    emp.empno%TYPE)  
  RETURN NUMBER  
IS  
  v_sal    emp.sal%TYPE := 0;  
BEGIN  
  SELECT sal INTO v_sal  
  FROM emp  
  WHERE empno=v_empno;  
  RETURN (v_sal);  
END get_sal;
```

**USANDO AS EXCEÇÕES DE RUNTIME**

Se uma função ou uma procedure não tem um tratamento de erros ou exceção, quando o erro ocorre a operação é abortada.

Para prevenir possíveis erros ou tratar as exceções que podem ocorrer devemos utilizar as EXCEPTION do Oracle ou definidas pelo usuário.

TIPO	TRATAMENTO	METODO UTILIZADO
ORACLE	Comunica o erro interativamente	É declarada na EXCEPTION
USUÁRIO	Comunica o erro interativamente	Faz a chamada a procedure RAISE_APPLICATION_ERROR
NÃO PREDEFINIDA ORACLE	Recebe o erro do banco ou customiza a mensagem de erro	É declarada como PRAGMA EXCEPTION_INIT

### Exceção Definida Pelo Usuário

No exemplo abaixo, é realizado um teste para verificar se o código do funcionário existe, se não existir a operação terminará com a mensagem definida pelo usuário.

```
CREATE OR REPLACE PROCEDURE FIRE_EMP
  ( v_empno IN    emp.empno%TYPE)
IS
BEGIN
  DELETE FROM emp
    WHERE empno=v_empno;
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR (-20200, 'Empregado não existe');
  END IF;
  COMMIT;
END fire_emp;

SQL> EXECUTE fire_emp (9999);
begin fire_emp (9999); end;

*
ERROR at line 1;
```

ORA-20200: Empregado não existe.

Nas exceções definidas pelo usuário, o número de erro pode ser definido dentro da faixa 20000 até 20999.



### Exceção definida pelo ORACLE

No exemplo abaixo, é realizado um teste para verificar se o código do chefe do funcionário existe, se não existir a operação terminará com a mensagem definida pelo usuário.

```
CREATE OR REPLACE PROCEDURE hire_emp
  ( v_ename IN emp.ename%TYPE,
    v_job    IN emp.job%TYPE,
    v_mgr    IN emp.mgr%TYPE,
    v_sal    IN emp.sal %TYPE)
IS
  v_hiredate emp.hiredate%TYPE;
  v_comm     emp.comm%TYPE;
  v_deptno   emp.deptno%TYPE;
BEGIN
  v_hiredate := sysdate;
  IF v_job = 'SALESMAN' THEN
    v_comm := 0;
  ELSE
    v_comm := NULL;
  END IF
  SELECT deptno INTO v_deptno FROM emp WHERE empno=v_mgr;
  INSERT INTO emp
  VALUES (v_empno, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno);
  COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR (-20201, 'Gerente não existe');
END hire_emp;

SQL> EXECUTE hire_emp ('JOE','CLERK', 9999,1000);
begin hire_emp ('JOE','CLERK', 9999,1000); end;
*
ERROR at line 1;
ORA-20201: Gerente não existe.
Exceção NÃO_PREDEFINIDA pelo ORACLE
```

No exemplo abaixo, é realizado um teste para verificar se o código do chefe do funcionário existe, se não existir a operação terminará com a mensagem definida pelo usuário.

```
CREATE OR REPLACE PROCEDURE hire_emp
  ( v_empno    IN emp.empno%TYPE,
    v_ename    IN emp.ename%TYPE,
    v_job      IN emp.job%TYPE,
    v_mgr      IN emp.mgr%TYPE,
    v_sal      IN emp.sal %TYPE)
  v_hiredate  IN emp.hiredate%TYPE;
  v_comm      IN emp.comm%TYPE;
  v_deptno    IN emp.deptno%TYPE;
IS
  e_invalid_manager  EXCEPTION;
  PRAGMA EXCEPTION_INIT (e_invalid_manager, -2291);
BEGIN
  INSERT INTO emp
    VALUES (v_empno, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno);
  COMMIT;
EXCEPTION
  WHEN e_invalid_manager THEN
    RAISE_APPLICATION_ERROR (-20201, 'Gerente não existe');
END hire_emp;

SQL> EXECUTE hire_emp (1234, 'JOE','CLERK', 9999,'01-jan-93', ->1000,null,20);
begin hire_emp (1234, 'JOE','CLERK', 9999,'01-jan-93', ->1000,null,20); end;
*
ERROR at line 1;
ORA-20201: Gerente não existe.
```

## EXECUTANDO PROCEDURES

Para executar uma procedure dentro do ambiente SQL\*Plus, utiliza-se o comando EXECUTE.

```
ACCEPT p_empno PROMPT "Entre com o numero do empregado"  
EXECUTE fire_emp (&p_empno);
```

Executando uma procedure de dentro de um programa em C

```
void run_fire_emp ()
```

```
{  
    EXEC SQL BEGIN DECLARE SECTION;  
        int empno;  
    EXEC SQL END DECLARE SECTION;  
    printf ("\n Entre com o numero do empregado : ");  
    scanf ("%d", &empno);  
    EXEC SQL EXECUTE;  
        begin  
            fire_emp (:empno);  
        end;  
    EXEC SQL END-EXEC;  
    return;  
}
```

### EXECUTANDO UMA PROCEDURE DE OUTRO ESQUEMA

```
SQL> EXECUTE scott.fire_emp (9999);
```

### EXECUTANDO UMA PROCEDURE DE UM BANCO REMOTO.

```
SQL> EXECUTE scott.fire_emp@fortal (9999);
```

### EXECUTANDO UMA PROCEDURE COM PASSAGEM POSICIONAL DE PARAMETROS.

```
SQL> EXECUTE fire_emp ('scott', 'analista', 9999);
```

### EXECUTANDO UMA PROCEDURE COM PASSAGEM NOMINAL DE PARAMETROS.

```
SQL> EXECUTE fire_emp (v_ename=>'scott', v_job=>'analista', v_sal=>9999);
```

### EXECUTANDO UMA PROCEDURE COM PASSAGEM NOMINAL E POSICIONAL DE PARAMETROS.

```
SQL> EXECUTE fire_emp (v_ename=>'scott', 'analista', 9999);
```

---

**EXECUTANDO FUNÇÕES**

---

**EXECUTANDO UMA FUNÇÃO DE DENTRO DE UM BLOCO PL/SQL ANONIMO.**

```
DECLARE
  v_empno NUMBER := 9876;
  v_sal    NUMBER := 0;
BEGIN
  ...
  v_sal := get_sal (v_empno);
  ...
END;
```

**EXECUTANDO UMA FUNÇÃO DE DENTRO DE UMA PROCEDURE.**

```
CREATE OR REPLACE PROCEDURE process_emp (v_empno IN emp.empno%TYPE)
IS
  v_sal    NUMBER := 0;
BEGIN
  ...
  v_sal := get_sal (v_empno);
  ...
END;
```

**EXECUTANDO UMA FUNÇÃO DENTRO DO AMBIENTE SQL\*PLUS.**

```
ACCEPT p_empno PROMPT "Entre com o numero do empregado"
VARIABLE g_sal NUMBER
EXECUTE :g_sal := fire_emp (&p_empno);
PRINT g_sal
```

**EXECUTANDO UMA FUNÇÃO DE DENTRO DE UM PROGRAMA EM C**

```
void run_fire_emp ()
{
    EXEC SQL BEGIN DECLARE SECTION;
        int empno;
        float sal;
    EXEC SQL END DECLARE SECTION;
    printf ("\n Entre com o numero do empregado : ");
    scanf ("%d", &empno);
    EXEC SQL EXECUTE;
        begin
            :sal := get_sal (:empno);
        end;
    EXEC SQL END-EXEC;
    printf ("\n O salario e: %7,2f", sal);
    return;
}
```

## DEBUG DE PROCEDURES E FUNÇÕES

Para verificar o conteúdo de uma procedure ou função, utiliza-se a VIEW USER\_SOURCE.

Para verificar os argumentos de uma função ou procedure, utilize o comando DESCRIBE.

Para verificar os erros de compilação da PROCEDURE ou FUNÇÃO, utilize a VIEW USER\_ERRORS ou simplesmente SHOW ERRORS.

CATEGORIA	DBMS_OUTPUT PROCEDURE	DESCRIÇÃO
SAIDA	PUT	Dá um display na execução da procedure
	NEW_LINE	Pula uma linha durante o DEBUG
	PUT_LINE	Combina os dois comandos anteriores
ENTRADA	GET_LINE	Recupera a linha corrente de uma saída de procedure
	GET_LINES	Recupera um array de linhas da saída do buffer de uma procedure
OUTROS	ENABLE	Habilita o DBMS_OUTPUT
	DISABLE	Desabilita o DBMS_OUTPUT

Para inicializar o DBMS\_OUTPUT dentro do SQL\*Plus ou no SQL\*DBA utiliza a opção de SERVEROUTPUT.

SET serveroutput ON

Exemplo de uso do DEBUG.

```
CREATE OR REPLACE FUNCTION average_sal
(v_n IN NUMBER)
RETURN NUMBER
IS
  CURSOR emp_cursor IS
    SELECT empno, sal FROM emp
    ORDER BY sal DESC;
  v_total_sal emp.sal%TYPE := 0;
  v_counter NUMBER;
BEGIN
  FOR r_emp IN emp_cursor LOOP
    EXIT WHEN emp_cursor%ROWCONT > v_n;
    v_total_sal := v_total_sal + r_emp.sal;
    v_counter := emp_cursor%ROWCONT;
    DBMS_OUTPUT.PUT ('Loop = ');
    DBMS_OUTPUT.PUT (v_counter);
    DBMS_OUTPUT.PUT ('; Empno = ');
    DBMS_OUTPUT.PUT (r_emp.empno);
    DBMS_OUTPUT.NEW_LINE;
```

```
END LOOP;
RETURN (v_total_sal/v_counter);
END average_sal;

SQL> SET verify OFF
SQL> SET serveroutput ON
SQL> DEFINE p_n=3
SQL> VARIABLE g_average NUMBER
SQL> EXECUTE :g_average := average_sal (&p_n);
Loop = 1; Empno = 7839
Loop = 2; Empno = 7788
Loop = 3; Empno = 7902

PL/SQL procedure successfully completed

SQL> PRINT g_average
G_AVERAGE
-----
36666.66667
```

# Desenvolvendo e Utilizando PACKAGES

## CRIAÇÃO DE PACKAGES (PACOTES)

Um pacote é dividido em duas partes, uma parte pública de especificação CREATE PACKAGE e outra privada ou do corpo CREATE PACKAGE BODY.

Em um pacote nós utilizamos Variáveis, Cursores, Constantes, Exceções, Procedures e Funções.

```
CREATE OR REPLACE PACKAGE package_name IS/AS
    variable_declaration
    cursor_declaration
    exception_declaration
    procedure_declaration
    function_declaration
END package_name
```

onde:

- **package\_name**  
Nome do pacote
- **variable\_declaration**  
Declaração de variáveis e constantes
- **cursor\_declaration**  
Declaração dos cursores explícitos
- **exception\_declaration**  
Declaração das exceções
- **procedure\_declaration**  
Declaração das procedures
- **function\_declaration**  
Declaração das funções

1. Utilize REPLACE se o pacote já existir;
2. Inicialize as variáveis, constantes ou formulas com algum valor, caso contrário terão valor NULL.



Exemplo:

```
CREATE OR REPLACE PACKAGE comm_package IS  
  g_comm_rate NUMBER := 0.1;  
  PROCEDURE reset_comm_rate  
    (v_comm_rate IN NUMBER);  
END comm_package;
```

**CRIAÇÃO DE PACKAGE BODY (CORPO DE PACOTES)**

```
CREATE OR REPLACE PACKAGE BODY package_name IS/AS
  variable_declaration
  cursor_declaration
  exception_declaration
  procedure_declaration
  function_declaration
END package_name
```

onde:

- **package\_name**  
Nome do pacote
- **variable\_declaration**  
Declaração de variáveis e constantes
- **cursor\_declaration**  
Declaração dos cursores explícitos
- **exception\_declaration**  
Declaração das exceções
- **procedure\_declaration**  
Declaração das procedures
- **function\_declaration**  
Declaração das funções

1. Utilize REPLACE se o pacote já existir;
2. Inicialize as variáveis, constantes ou formulas com algum valor, caso contrário terão valor NULL.

Exemplo:

```
CREATE OR REPLACE PACKAGE BODY comm_package IS
  FUNCTION validate_comm_rate
    (v_comm_rate IN NUMBER)
    RETURN BOOLEAN
  IS
    v_max_comm_rate NUMBER;
  BEGIN
    SELECT MAX(comm/sal) INTO v_max_comm_rate
    FROM EMP;
    IF v_comm_rate > v_max_comm_rate THEN
      RETURN FALSE;
    
```

```
        ELSE
            RETURN TRUE;
        END IF;
    END validate_comm_rate;

    PROCEDURE reset_comm_rate
        (v_comm_rate IN NUMBER)
    IS
        v_valid    BOOLEAN;
    BEGIN
        v_valid := validate_comm_rate (v_comm_rate);
        IF v_valid = TRUE THEN
            :g_comm_rate := v_comm_rate;
        ELSE
            RAISE_APPLICATION_ERROR (-20210, 'Rateio invalido de comissão');
        END IF;
    END reset_comm_rate;
END comm_package;
```

Se necessitar que seja executada uma rotina somente na 1ª chamada do pacote para cada sessão, adicione um bloco PL/SQL no PACKAGE BODY, da seguinte forma.

```
CREATE OR REPLACE PACKAGE BODY package_name IS
    declarações necessárias de variáveis, funções, procedures.
BEGIN
    SELECT AVG(COMM/SAL) INTO G_COMM_RATE FROM EMP;
END package_name;
```

## EXECUTANDO PACKAGES

A execução de um pacote segue a mesma regra das chamadas de procedures.

Para executar uma procedure de um pacote de dentro do ambiente SQL\*Plus, utiliza-se o comando EXECUTE.

```
SQL> EXECUTE comm_package.reset_comm_rate (.15);
```

### EXECUTANDO UMA PROCEDURE DE UM PACOTE DE OUTRO ESQUEMA

```
SQL> EXECUTE scott.comm_package.reset_comm_rate (.15);
```

### EXECUTANDO UMA PROCEDURE DE UM PACOTE DE UM BANCO REMOTO.

```
SQL> EXECUTE scott.comm_package.reset_comm_rate@fortal (.15);
```

## RECOMPILANDO PACKAGES

Para recompilar pacotes devemos utilizar os seguintes comandos:

### RECOMPILAÇÃO DA ESPECIFICAÇÃO DO PACOTE E DO CORPO

```
ALTER PACKAGE package_name COMPILE
```

### RECOMPILAÇÃO SOMENTE DA ESPECIFICAÇÃO

```
ALTER PACKAGE package_name COMPILE PACKAGE
```

### RECOMPILAÇÃO SOMENTE DO CORPO

```
ALTER PACKAGE package_name COMPILE BODY
```

# **Desenvolvendo DATABASE TRIGGERS**

Antes de criar um trigger devemos decidir quanto a tempo, evento, tipo e o corpo do trigger.

PARTE	DESCRIÇÃO	VALORES POSSÍVEIS
TEMPO	Quando o trigger vai ser disparado com relação ao evento	BEFORE AFTER
EVENTO	Operação realizada na tabela que causa a execução do trigger.	INSERT UPDATE DELETE
TIPO	Quantas vezes o corpo do trigger sera executado.	STATEMENT ROW
CORPO	Qual a ação que o trigger executara.	BLOCO PL/SQL

Um máximo de 12 triggers podem ser criados para uma mesma tabela a partir da combinação dos componentes acima.

## **CRIANDO TRIGGERS**

A sintaxe para criação de triggers é:

```
CREATE [OR REPLACE] TRIGGER [schema.]trigger
  {BEFORE | AFTER}
  {DELETE | INSERT | UPDATE [OF column [, column] ...]}
  [OR {DELETE | INSERT | UPDATE [OF column [, column] ...]}] ...
  ON [schema.]table
  FOR EACH ROW
  [WHEN (condition)] ]
  pl/sql_block
```

---

**APLICANDO TRIGGERS NO CONTROLE DA SEGURANÇA**

---

Exemplo de trigger BEFORE STATEMENT, para restringir a inserção de dados na tabela funcionarios para somente dias uteis da semana.

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON emp
BEGIN
  IF (TO_CHAR(sysdate,'DY') IN ('SAT','SUN'))
    OR (TO_NUMBER(sysdate,'HH24') NOT BETWEEN 8 AND 18) THEN
    RAISE_APPLICATION_ERROR (-20500, 'A inserção de dados na tabela de funcionários só é
possivel durante o horario normal de trabalho');
  END IF;
END;
```

Os triggers que usam BEFORE são muito usados para inicializar variaveis globais ou flags e para validar regras complexas de negocio.

**APLICANDO TRIGGERS PARA AUDITAR VALORES**

Exemplo de trigger AFTER STATEMENT, para restringir a alteração de dados na tabela funcionarios para somente a quantidade de alterações cadastradas para cada funcionário.

```
CREATE OR REPLACE TRIGGER check_sal_count
AFTER UPDATE OF sal ON emp
DECLARE
    v_sal_changes NUMBER;
    v_max_changes NUMBER;
BEGIN
    SELECT upd, max_upd INTO v_sal_changes, v_max_changes
    FROM audit_table
    WHERE user_name = user
    AND table_name = 'EMP'
    AND column_name = 'SAL';
    IF v_sal_changes > v_max_changes THEN
        RAISE_APPLICATION_ERROR (-20501,
        'Voce tem um maximo de ' || TO_CHAR(v_max_changes) || 'para a coluna SAL');
    END IF;
END;
```

---

**APLICANDO TRIGGERS COM VÁRIOS EVENTOS**

---

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE DELETE OR INSERT OR UPDATE ON emp
BEGIN
  IF (TO_CHAR(sysdate,'DY') IN ('SAT','SUN'))
    OR (TO_NUMBER(sysdate, 'HH24') NOT BETWEEN 8 AND 18) THEN
    IF DELETING THEN
      RAISE_APPLICATION_ERROR (-20500,
'A exclusão de dados na tabela de funcionários é somente no horario normal');
    ELSIF INSERTING THEN
      RAISE_APPLICATION_ERROR (-20501,
'A inclusão de dados na tabela de funcionários é somente no horario normal');
    ELSIF UPDATING THEN
      RAISE_APPLICATION_ERROR (-20502,
'A alteração de dados na tabela de funcionários é somente no horario normal');
    END IF;
  END IF;
END;
```



---

**APLICANDO TRIGGERS DE AÇÃO POR LINHA**

---

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OF sal ON emp
FOR EACH ROW
WHEN NEW.job = 'SALESMAN
BEGIN
    :new.comm := :old.comm * (:new.sal / :old.sal);
END;
```

A cláusula WHEN não pode ser utilizada em STATEMENT trigger.

**APLICANDO TRIGGERS PARA REPLICAR TABELAS**

```
CREATE OR REPLACE TRIGGER emp_replica
BEFORE INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    IF :new.flag IS NULL THEN
      INSERT INTO emp@fortal
        VALUES (:new.empno, :new.ename,...);
      :new.flag = 'A';
    END IF;
  ELSE
    IF :new.flag = :old.flag THEN
      UPDATE emp@fortal
        SET   ename = :new.ename;
             FLAG = :new.flag ;
      WHERE empno = :new.empno;
      IF :old.flag = 'A' THEN
        :new.flag = 'B';
      ELSE
        :new.flag = 'A';
      END IF;
    END IF;
  END IF;
END;
```

---

## GERENCIANDO E ARMAZENANDO OS DATABASE TRIGGERS

---

- Para criar um novo trigger utilize o comando CREATE TRIGGER
- Para modificar um trigger existente utilize o comando CREATE OR REPLACE TRIGGER;
- Para remover um trigger utilize o comando DROP TRIGGER
- Para desabilitar um trigger utilize o comando:
  - ALTER TRIGGER trigger\_name DISABLE
- Para reabilitar um trigger utilize o comando:
  - ALTER TRIGGER trigger\_name ENABLE
- Para desabilitar todos os trigger de uma tabela utilize o comando:
  - ALTER TRIGGER table\_name DISABLE ALL TRIGGERS
- Para reabilitar todos os trigger de uma tabela utilize o comando:
  - ALTER TRIGGER table\_name ENABLE ALL TRIGGERS
- Para verificar o conteudo de um bloco PL/SQL dentro do trigger, utilize a VIEW USER\_TRIGGERS.

[illegible]

# Apêndice 1

## **Exercícios**

1. Selecione todas as informações da tabela de salários (SALGRADE).
2. Selecione todas as informações da tabela de empregados (EMP).
3. Liste todos os empregados que possuem salário entre 1000 e 2000, da tabela EMP.
4. Liste o numero e o nome do departamento em ordem de nome (DEPT).
5. Liste todos os diferentes tipos de funções, sem repetições,
6. Liste todos os dados dos empregados do departamento 20 em ordem alfabética de nome.
7. Liste nome e função de todos os SALESMAN do departamento 20.
8. Liste todos os empregados que possuem no nome 'LL' e 'TH'.
9. Liste todos os empregados que possuem um gerente (MGR).
10. Liste o nome e a remuneração anual de todos os empregados.
11. Liste os empregados com data de admissão no ano de 1983, concatene o número com o nome do empregado separados por hífen.
12. Liste todos os dados da tabela EMP, classificando-os por MGR. Se MGR igual a nulo coloque no final da lista.
13. Liste o departamento, nome e salário dos empregados selecionando a função no momento da execução do comando.
14. Liste a data atual, adicione 2 meses, e mostre a próxima sexta-feira daquela semana, edite esta data no formato 'DD/MM/YY'.
15. Pesquise a tabela EMP e mostre o nome do funcionário que não tem um superior, colocando 'SEM CHEFE' para ele.
16. Liste nome e data de admissão dos funcionários do departamento 20, coloque um ALIAS para data de admissão e mostre no formato (FMMONTH, DDSPTH YYYY).
17. Liste o nome, data de admissão e data de aumento salarial considerando 1 ano após a admissão. Ordene a saída pela ascendente do aumento.
18. Liste o nome, o salário dos funcionários. Se o salário for maior que 1500, mostre-o, se for igual a 1500 mostre 'EXATO', se for menor mostre 'ABAIXO DE 1500'.
19. Escreva uma pesquisa que retorne o dia da semana para alguma data informada no formato: 'DD/MM/YY'.
20. Liste o departamento, nome e salário dos empregados incrementando em 15%, mostrando somente a parte inteira do valor.
21. Selecionar Código, Nome, Salário e Comissão dos funcionários que tem comissão maior que 5% do salário.
22. Selecionar os funcionários que tem as funções de CLERK ou ANALYST
23. Selecionar os nomes dos funcionários que tenham a letra "E" na segunda posição do nome;
24. Selecionar os funcionários que trabalhem no departamento 10 e tenham salário superior a 1100.
25. Selecionar os funcionários admitidos após 01/01/88.
26. Selecionar os funcionários que tem cargo de MANAGER ou que trabalhem no departamento 10 e sejam CLERK.
27. Selecionar os funcionários que não tem comissão.

28. Selecionar os funcionários com seus respectivos salários, comissão e a soma de ambos. Ordenar a saída por Departamento e Total.
29. Selecionar os funcionários e seus locais de trabalho.
30. Selecionar os funcionários e seus respectivos departamentos incluindo os departamentos sem funcionários.
31. Selecionar o nome do funcionário e o nome do seu superior.
32. Selecionar os funcionários que ganham mais que o funcionário MILLER.
33. Selecionar o nome, salário comissão sendo que em cada linha deverá conter somente um tipo de rendimento (salário ou comissão) e este deverá estar indicado com tipo correspondente.
34. Selecionar os códigos de departamentos que existam simultaneamente nas tabelas EMP e DEPT.
35. Selecionar os nomes dos funcionários que tenham a mesma função de ADAMS.
36. Selecionar os funcionários que trabalham em DALLAS e que tenham a mesma função de JONES.
37. Selecionar os funcionários que ganham mais que todos os funcionários do departamento 30.
38. Selecionar nome e função dos funcionários que tenham a mesma função e salário de ADAMS.
39. Selecionar o nome e função dos funcionários que trabalham no departamento 10 e que tenham as mesmas funções do departamento RESEARCH.
40. Selecionar nome e função dos funcionários, sendo que para as funções de MANAGER e SALESMAN deverão retornar os códigos 1 e 2 respectivamente. Para as demais funções deverá retornar 3.
41. Para o código do departamento 10, você devera retornar a função do funcionário, e para os demais o nome do mesmo.
42. Selecionar os funcionários, calculando o seu salário diário com arredondamento de duas casas decimais, considerar o mês útil de 22 dias.
43. Selecionar nome e função dos funcionários, sendo que ambas deverão estar em uma única coluna e estes devem estar separados por uma “/”.
44. Localiza a posição da 2ª. ocorrência do caracter “E” nas funções da tabela EMP.
45. Selecionar o nome e as iniciais de sua função, considerando que as iniciais são constituídas pelas 3 primeiras letras da função.
46. Selecionar as funções existentes e o seu tamanho.
47. Selecionar nome e função dos funcionários, sendo que as funções devem ser apresentadas em minúsculas.
48. Selecionar nome e função dos funcionários, sendo que as funções devem ser apresentadas com a 1ª. letra em maiúsculas.
49. Selecionar os nomes de funcionários, sendo que para os nomes que começam com a letra “A” deverão ter este caracter removido.
50. Remover os caracteres a esquerda do nome de funcionário que pertença ao conjunto de caracteres “LL”.
51. Apresentar a soma dos salários do departamento 20.
52. Selecionar o funcionário que tem o maior salário da companhia.
53. Selecionar o numero de funcionários que tem comissão.
54. Selecionar o numero de funções existentes.

55. Selecionar os departamentos que tem o maior e o menor salário da companhia.
56. Selecionar as funções existentes, o seu numero e a média salarial por função.
57. Selecionar as funções existentes, o seu numero e a média salarial por função, mas somente dos cargos que tem média salarial superior a 1500.
58. Selecionar o departamento que tem o maior numero de funcionários.
59. Selecionar as funções existentes, o seu numero e a média salarial por função, mas somente para as funções que tem mais de uma pessoa.
60. Selecionar o nome dos funcionários, apresentando uma estrutura hierárquica, ou seja, o organograma, indentando os nomes de 2 caracteres quando houver mudança de nível.
61. Mostre o funcionário que ganha menos na empresa.
62. Mostre o funcionário que ganha mais em cada departamento da empresa.
63. Quais são os cargos que existem no departamento 10 e 30.
64. Quais são os cargos que existem no departamento 10 que não existem no 30.
65. Selecionar os funcionários que ganham mais que todos os funcionários do departamento 10 e que ganham acima da media do departamento 20.
66. Listar a media salarial de cada função que não exista no departamento 10.
67. Listar a media mensal dos salários para cada função dentro do departamento.
68. Selecione a 1ª. ocorrência da letra "L" do nome do empregado e troque-o por "X".
69. Selecionar as 10 primeiras linhas da tabela EMP. A numeração deverá ser mostrada, precedendo a linha.
70. Criar uma tabela com o nome de "CURSO" com as seguintes colunas:

COLUNA	TAMANHO	TIPO	OBRIGATORIO
NOME	30	VARCHAR2	SIM
CPF_CGC	14	NUMERICO	SIM
ENDereco	20	VARCHAR2	NÃO
TELEFONE	11	NUMERICO	NÃO
DATA_NASC		DATA	NAO

71. Altere a tabela criada no exercício anterior para que o nome tenha 40 posições.
72. Adicionar uma coluna a tabela CURSO, com a seguinte especificação: PROFISSAO, TAMANHO 15, VARCHAR2, NÃO OBRIGATÓRIO.
73. Criar uma tabela chamada EXEMPLO2, a partir das colunas NOME e CARGO da tabela EMP.



74. Criar uma tabela chamada FUNC, contendo as seguintes colunas:

COLUNA	TAMANHO	TIPO	OBRIGATORIO
CODIGO_FUNCIONARIO	4	NUMERICO	SIM
NOME_FUNCIONARIO	20	VARCHAR2	NÃO
ADMISSAO		DATA	NÃO
TELEFONE	11	NUMERICO	NÃO
DATA_NASC		DATA	NAO

75. Criar um índice que garanta a unicidade sobre o código do funcionário.

76. Criar uma seqüência com o nome CODFUNC que gere o código do funcionário iniciando por 1000 e cresça na ordem de uma unidade.

77. Inserir 5 funcionários na tabela FUNC, utilizando a seqüência CODFUNC.

78. Inserir na tabela FUNC, todos os funcionários da tabela EMP, sendo que o código do funcionário deverá ser da seqüência CODFUNC.

79. Liste o nome do departamento centralizado, assumindo 20 para o tamanho da coluna.

80. Liste o nome do empregado e a sua função concatenada, sendo que o nome deve ser alinhado pela esquerda e a função pela direita.

# *Apêndice 2*

## **RESPOSTAS DOS EXERCÍCIOS**

1.    select \* from salgrade  
      /
2.    select \* from emp  
      /
3.    select \* from emp  
      where sal >= 1000 and  
              sal <= 2000  
      /
4.    select \* from dept  
      order by dname  
      /
5.    select distinct job from emp  
      /
6.    select \* from emp  
      where deptno = 20  
      order by ename  
      /
7.    select ename nome, MGR funcao from emp  
      where job = 'SALESMAN' AND  
              DEPTNO = 20  
      /
8.    SELECT \* FROM EMP  
      WHERE ENAME LIKE '%LL%' AND ENAME LIKE '%TH%'  
      /
9.    SELECT \* FROM EMP  
      WHERE MGR IS NOT NULL  
      /
10.   SELECT ENAME NOME, SAL\*12 + NVL(COMM,0) "REMUNERACAO" FROM EMP  
      /
11.   SELECT EMPNO||'-'||ENAME FROM EMP  
      WHERE TO\_CHAR(HIREDATE, 'YYYY')=1983  
      /
12.   SELECT \* FROM EMP  
      ORDER BY MGR  
      /
13.   SELECT DEPTNO, ENAME, SAL FROM EMP  
      WHERE upper(JOB) = UPPER('&FUNCAO')  
      /
14.   select sysdate data1,  
          add\_months(sysdate,2) data2,  
          TO\_CHAR(next\_day(add\_months(sysdate,2), 'FRIDAY'), 'DD/MM/YY') DATA3 from  
      dual  
      /

15. 

```
select ename || '-' || 'sem chefia' from emp
where mgr is null
/
```
16. 

```
select ename, to_char(hiredate, 'fmmonth, ddsptth yyyy') "data admissao"
from emp
where deptno = 20
/
```
17. 

```
select ename, to_char(hiredate, 'dd/mm/yy'), add_months(hiredate, 12)
"dt aumento" from emp order by "dt aumento"
/
```
18. 

```
select ename, decode(sign(sal-1500), 0, 'exato', 1, sal, -1, 'Menor')
from emp
/
```
19. 

```
select decode(to_char(to_date('&data', 'dd/mm/yy'), 'D'),
1, 'Dom', 2, 'Seg', 3, 'Ter', 4, 'Qua', 5, 'Qui', 6, 'Sex', 7, 'Sab') from dual
/
```
20. 

```
select deptno, ename, sal, trunc(sal*1.15) from emp
/
```
21. 

```
select empno, ename, sal, comm from emp
where comm > (sal*0.05)
/
```
22. 

```
select ename from emp
where upper(job)=upper('clerk') or
upper(job)=upper('analyst')
/
```
23. 

```
select ename from emp
where ename like '_E%'/
/
```
24. 

```
select ename, sal from emp
where deptno=10 and sal > 1100
/
```
25. 

```
select ename, hiredate from emp
where hiredate > to_date('01/01/88', 'dd/mm/yy')
/
```
26. 

```
select ename, deptno, job from emp
where upper(job)=upper('manager') or
(deptno=10 and upper(job)=upper('clerk'))
/
```
27. 

```
select ename from emp where nvl(comm, 0) = 0
/
```
28. 

```
select deptno, ename, sal, comm, sal+nvl(comm, 0) total from emp
order by deptno, total
/
```

29. 

```
select ename, loc from emp e, dept d
      where e.deptno=d.deptno
      /
```
30. 

```
select ename,dname from emp e, dept d
      where e.deptno(+)=d.deptno
      /
```
31. 

```
select e.ename,e.job,m.ename,m.job from emp e, emp m
      where e.mgr=m.empno
      /
```
32. 

```
select ename,sal from emp
      where sal > (select sal from emp
                  where ename='MILLER')
      /
```
33. 

```
select ename,'Sal',sal from emp
      union
      select ename,'com',comm from emp
      where comm is not null
      /
```
34. 

```
select deptno from emp
      intersect
      select deptno from dept
      /
```
35. 

```
select ename from emp
      where job=(select job from emp where ename='ADAMS') and
      ename <> 'ADAMS'
      /
```
36. 

```
select e.ename, e.job, d.loc from emp e, dept d
      where d.loc='DALLAS' and e.deptno=d.deptno and
      e.job = (select job from emp
              where ename='JONES') and
      ename <> 'JONES'
      /
```
37. 

```
select ename from emp
      where sal > all (select distinct sal from emp
                     where deptno=30)
      /
```
38. 

```
select ename,job,sal from emp
      where (job,sal)=(select job,sal from emp where ename='ADAMS') and
      ename <> 'ADAMS'
      /
```
39. 

```
select ename,job from emp e
      where e.deptno=10 and
      job in (select distinct job from emp f
             where f.deptno=(select d.deptno from dept d
                             where d.dname='RESEARCH'))
      /
```
40. 

```
select ename,JOB,decode(job, 'MANAGER',1, 'SALESMAN',2,3) CODIGO FROM EMP
```

```

/
41. select ename,job,deptno,decode(sign(deptno-10),0,job,1,ename) from emp
/
42. select ename,sal,round(sal/22,2) from emp
/
43. select ename || '/' || job from emp
/
44. select job,instr(job,'E',1,2) from emp
/
45. select ename, job, substr(job,1,3) from emp
/
46. select ename,length(rtrim(job)) from emp
/
47. select ename, job, lower(job) from emp
/
48. select ename, job,initcap(job) from emp
/
49. select ename, ltrim(ename,'A') from emp
/
50. select ename, replace(ename,'LL') from emp
/
51. select deptno,sum(sal) from emp
    group by deptno
    having deptno=20
/
52. select e.ename, e.sal from emp e
    where e.sal = (select max(sal) from emp)
/
53. select count(nvl(comm,0)) from emp
    where comm is not null
/
54. select count(distinct job) from emp
/
55. select max(sal),min(sal) from emp
/
56. select job, avg(sal), count(job) from emp
    group by job
/
57. Select job, avg(sal), count(job) from emp
    having avg(sal) > 1500
    group by job

```

```

/
58. select dname from dept
    where deptno=(select deptno from emp
        group by deptno
        having count(empno)=(select max(count(empno)) from emp
            group by deptno))
/
59. select e.job,count(e.job),avg(e.sal) from emp e
    group by e.job
    having count(e.job) in (select count(job) from emp
        group by job) and
        count(e.job) > 1
/
60. select level,(lpad(' ',2*(level-1)))|| ename nome
    from emp start with job='PRESIDENT'
    connect by prior empno=mgr
/
61. Select e.ename,e.sal from emp e
    where e.sal = (select min(c.sal) from emp c)
/
62. select e.deptno,e.ename,e.sal from emp e
    where (e.deptno,e.sal) in (select c.deptno,max(c.sal) from emp c
        group by c.deptno)
/
63. select distinct job from emp
    where deptno in(10,30)
/
64. select e.job from emp e
    where e.deptno=10 and
        e.job<> all(select distinct job from emp
            where deptno=30)
/
65. select e.ename,e.sal from emp e
    where e.sal > all (select f.sal from emp f
        where f.deptno=10) and
        e.sal > (select avg(g.sal) from emp g
            where g.deptno=20)
/
66. select e.job,avg(e.sal) from emp e
    group by e.job
    having e.job <> all (select f.job from emp f
        where f.deptno=10)
/
67. select deptno,job,avg(sal) from emp
    group by deptno,job
/

```

```

68.  select ename, substr(ename,1,instr(ename,'L',1,1)-1)
      ||'X' || substr(ename,instr(ename,'L',1,1)+1)  from emp
      where instr(ename,'L',1,1)-1 > 0
      /

69.  select rownum, empno, ename, job, sal from emp where rownum <= 10
      /

70.  create table curso
      (nome          varchar2(30) not null,
       cpf_cgc       number(14)   not null,
       endereco       varchar2(20),
       telefone       number(11),
       data_nasc      date)
      /

71.  alter table curso
      MODIFY (nome varchar2(40))
      /

72.  alter table curso
      ADD (PROFISSAO varchar2(15))
      /

73.  create table exemplo2
      as select ename, job from emp
      /

74.  create table func
      (codigo_funcionario number(04) not null,
       nome_funcionario   varchar2(20),
       admissao           date,
       telefone           number(11),
       data_nasc          date)
      /

75.  create unique index uk_codfunc
      on func
      (codigo_funcionario)
      /

76.  create sequence codfunc
      increment by 1
      start with 1000
      nocache
      /

77.  insert into func
      values(codfunc.nextval, 'FRANCA', '01/APR/85',
            2879845, '10/NOV/63')
      /

78.  INSERT INTO FUNC(CODIGO_FUNCIONARIO,NOME_FUNCIONARIO)
      SELECT CODFUNC.NEXTVAL,ENAME FROM EMP
      /

79.  Select DNAME,
      LPAD(' ', ((20-length(RTRIM(Dname)))/2), ' ') ||

```



```
        RTRIM(DNAME) N  
    from DEPT  
/
```

```
80.  Select rpad(ename,20)||lpad(job,10) from emp  
/
```