

Módulo I

- ⇒ Sistema Gerenciador de Banco de Dados
- ⇒ Uma Visão Geral dos Produtos ORACLE

Introdução ao

Oracle7

Sistema Gerenciador de Banco de Dados

- **Objetivos desta Unidade:**
 - Esta unidade descreve conceitos de Bancos de Dados Relacionais tendo como enfoque o **ORACLE**.
- **O que é uma Base de Dados (Database)?**
 - Uma base de dados é uma coleção de informações organizadas.
 - Em um sistema manual tradicional os dados são geralmente guardados em fichas e estocados em arquivos. Para acessar estes dados, um acesso manual nestes arquivos é necessário.
 - Num sistema baseado em computador, os dados são armazenados tipicamente em discos e são acessados através de programas.

Sistema Gerenciador de Banco de Dados

- **Base de dados computadorizada:**

- Em uma base de dados computadorizada existem muitas vantagens:

- A alteração e a recuperação dos dados é mais rápida;
- A informação ocupa menos espaço para ser armazenada;
- Muitos usuários podem compartilhar as mesmas informações ao mesmo tempo; a redundância de dados é minimizada;
- Inconsistências podem ser evitadas;
- Padrões podem ser estabelecidos;
- Pode-se implementar níveis de segurança nestes dados;
- etc.

Sistema Gerenciador de Banco de Dados

- **O Database Management System (DBMS):**
 - Para controlar o acesso e o armazenamento das informações é necessário um DBMS. O DBMS (Sistema Gerenciador de Banco de Dados) é o programa que fica responsável pelo controle de acesso dos usuários à base de dados. Ele controla o armazenamento dos dados, a recuperação e a alteração nestes dados.
 - O DBMS age na interface entre a base de dados e os usuários da base de dados.
 - Dentre os tipos de bancos de dados existentes, os mais recentes são os bancos de dados relacionais, entre eles o **ORACLE**.

Sistema Gerenciador de Banco de Dados

- **Conceitos Relacionais:**

- Uma **base de dados relacional** é conhecida pelos usuários como uma coleção de tabelas de duas dimensões. Existem quatro conceitos básicos:
 - tabelas
 - colunas
 - linhas
 - campos
- O modelo relacional tem como base o ramo da matemática conhecido como álgebra relacional. Este modelo envolve:
 - uma coleção de objetos conhecidos como relações,
 - um conjunto de operadores que agem nestas relações produzindo novas relações.

Sistema Gerenciador de Banco de Dados

– Operadores Relacionais:

Relação	Descrição
Restrição	A restrição traz um subconjunto de linhas de uma tabela que atende a determinada restrição imposta. (Subconjunto Horizontal);
Projeção	Operação que mostra um subconjunto de colunas de uma tabela. (subconjunto vertical)
Produto	É o resultado do produto entre dois conjuntos de linhas de tabelas.
Join	É o resultado da combinação de dois conjuntos de dados sob determinadas condições.
Union	Tem como resultado todas as linhas que aparecem em ambas as relações.
Interseção	Tem como resultado somente as linhas que aparecem em ambas as relações
Diferença	Tem como resultado as linhas que aparecem em uma das relações mas não na outra.

Sistema Gerenciador de Banco de Dados

- **Propriedades de uma base de dados relacional.**
 - Uma base de dados relacional aparece como uma coleção de tabelas relacionadas para o usuário
 - Não existem pointers explícitos para os dados. O acesso é feito baseado no próprio dado.
 - A linguagem de acesso é não-procedural e english-like.
 - O usuário não especifica a rota de acesso aos dados e não precisa saber como os dados estão arranjados fisicamente.
 - Os comandos de acesso aos dados são feitos através da Linguagem SQL.

Uma Visão Geral dos Produtos ORACLE

- **ORACLE Server**

É o servidor do banco de dados, que gerencia o armazenamento e recuperação dos dados. Os demais produtos funcionam tendo o servidor como base.

- **SQL**

É a linguagem padrão dos bancos de dados relacionais, entre eles o ORACLE.

- **PL/SQL**

Extensões procedurais do ORACLE ao SQL.

Uma Visão Geral dos Produtos ORACLE

- **SQL*Plus**

É um ambiente através do qual os comandos SQL podem ser entrados e executados.

- **SQL*DBA**

Conjunto de ferramentas do administrador do banco de dados.

- **SQL*Loader**

Permite entrada de dados a partir de tabelas ASCII para as tabelas ORACLE

Uma Visão Geral dos Produtos ORACLE

- **Developer/2000**

Conjunto de ferramentas de desenvolvimento visual que permitem a criação de aplicativos de banco de dados.

- **Designer/2000**

Conjunto de ferramentas CASE que abrange todas as fases do desenvolvimento de aplicativos de banco de dados.

- **Data Query**

Ferramenta de consulta baseada em formulários para usuários finais.

- **Data Browser**

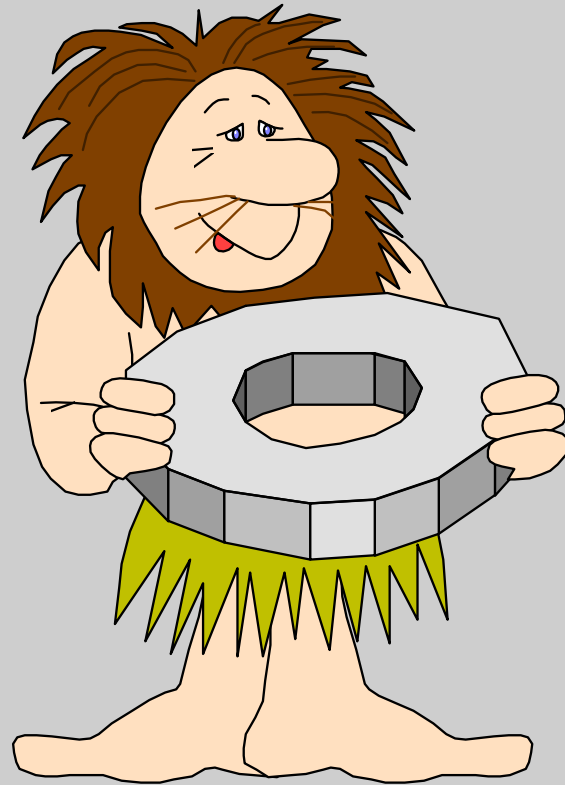
Ferramenta de consulta gráfica para usuários finais.

Módulo II

- ⇒ Introdução à Linguagem SQL
- ⇒ Escrevendo Comandos SQL
- ⇒ Variáveis de Substituição em Queries

Introdução ao

Oracle7



Introdução a Linguagem SQL

Introdução a Linguagem SQL

- **Objetivos desta unidade:**

- Esta unidade dá uma introdução à Linguagem SQL (**Structured Query Language**) usada para acessar uma base de dados Oracle.
- Veremos os comandos SQL necessários para:
 - executar cálculos;
 - trabalhar com valores nulos corretamente;
 - especificar cabeçalhos alternativos de colunas;
 - concatenar colunas;
 - mostrar colunas em determinada ordem.

- **Visão geral do SQL:**

- Para acesso a um banco de dados relacional, é necessário uma linguagem. O SQL é a linguagem usada pela maioria dos banco de dados relacionais.

Introdução a Linguagem SQL

- **Características do SQL:**

- O **SQL** é uma linguagem baseada no inglês, e usa palavras como: select, insert, delete como parte de seu conjunto de comandos;
- **SQL** é uma linguagem não-procedural; você especifica qual informação você quer e não como trazê-la. Em outras palavras, você não especifica qual vai ser o método de acesso aos dados. Todos os comandos SQL utilizam o **optimizer** que determina a maneira mais rápida de recuperar os dados.
- O **SQL** processa um conjunto de linhas por vez, ao invés de uma linha.
- O **SQL** oferece uma série de comandos para uma variedade de tarefas diferentes, incluindo:
 - seleção de dados;
 - inserção, alteração, e deleção de linhas em uma tabela;
 - criar, deletar e alterar objetos do banco de dados;
 - controlar o acesso aos dados e aos objetos do bando de dados;
 - garantir a consistência da base de dados;
 - etc.

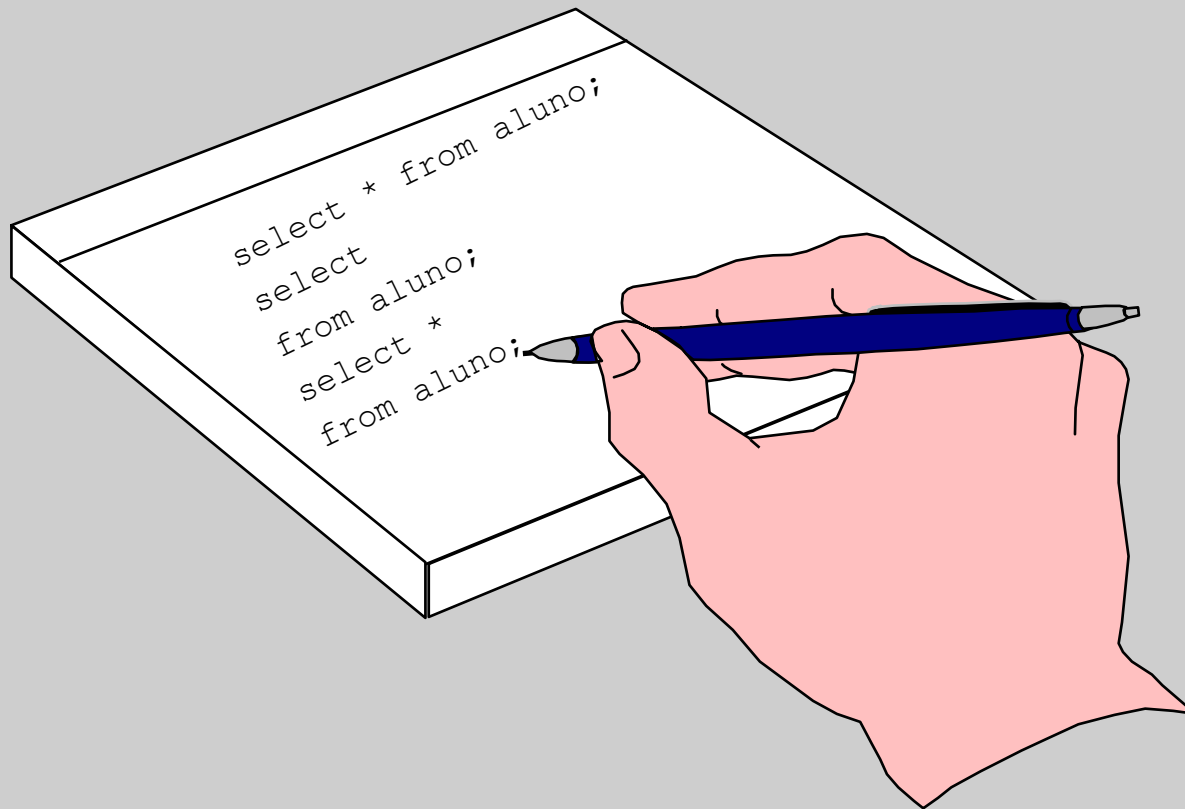
Introdução a Linguagem SQL

- **SQL*Plus:**

- O **SQL*Plus** é um ambiente através do qual os comandos SQL podem ser entrados e executados. Ele contém uma série de comandos que ajudam na edição salvamento e execução de comandos SQL ou arquivos de comandos SQL: **scripts**.

- **Principais comandos SQL:**

Comando	Descrição
select	seleciona dados de uma base de dados
insert	Insere linhas em uma tabela
update	altera valores de linhas na base de dados
delete	elimina linhas na tabela
create	cria objetos na base de dados
alter	altera a estrutura de um objeto da base
drop	elimina determinado objeto da base de dados
grant	dá direitos de acessos aos objetos do banco de dados
revoke	retira direitos de acesso aos objetos do banco



Escrevendo Comandos SQL

Escrevendo Comandos SQL

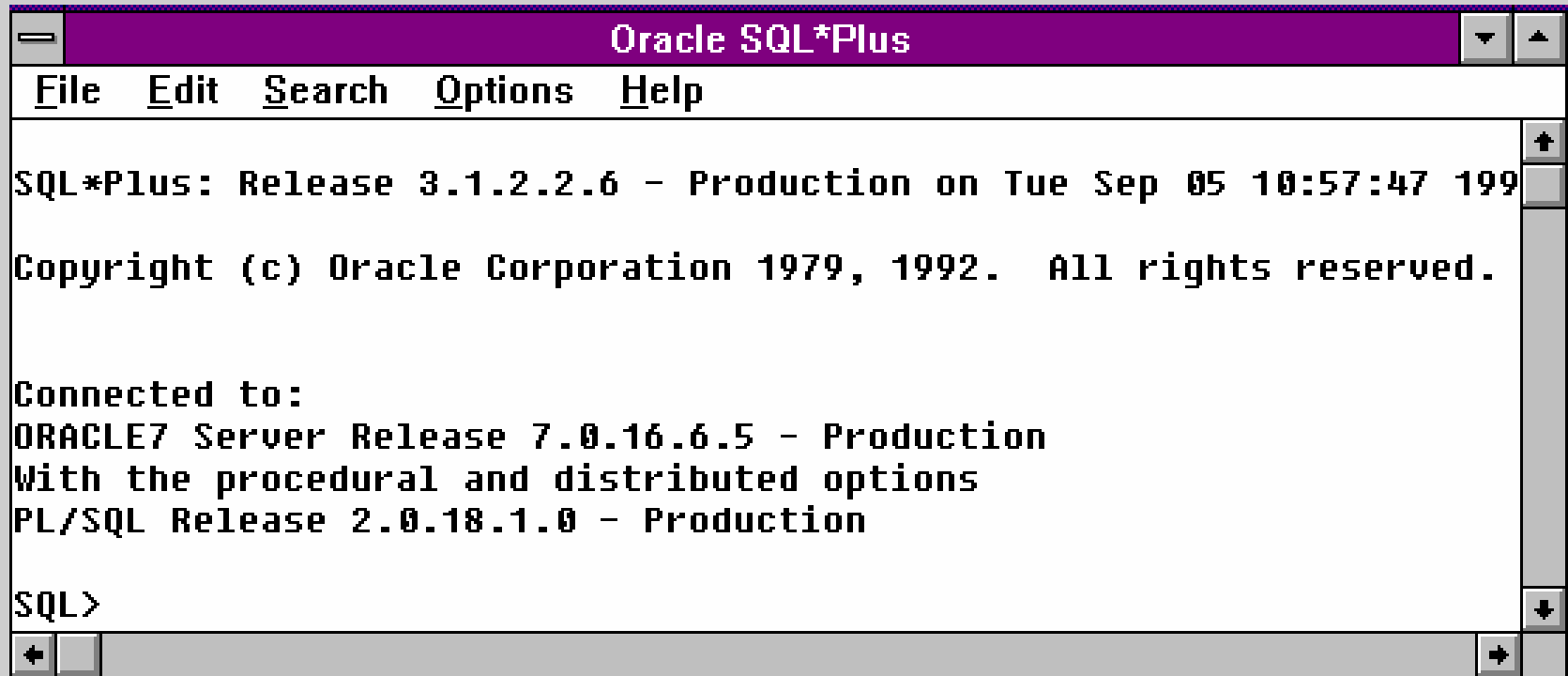
- Para escrever comandos SQL você precisa:
 - Estar no SQL*Plus.
 - Entre no SQL*Plus através de um duplo click no ícone:



- Será aberta uma caixa de diálogo para você entrar com o nome do seu usuário, sua password e a identificação do banco de dados que você vai entrar (connect string);

Escrevendo Comandos SQL

- A seguir será aberta a tela do **SQL*Plus** para você entrar com os comandos:



```
Oracle SQL*Plus
File Edit Search Options Help
SQL*Plus: Release 3.1.2.2.6 - Production on Tue Sep 05 10:57:47 199
Copyright (c) Oracle Corporation 1979, 1992. All rights reserved.

Connected to:
ORACLE7 Server Release 7.0.16.6.5 - Production
With the procedural and distributed options
PL/SQL Release 2.0.18.1.0 - Production

SQL>
```

Escrevendo Comandos SQL

- **Regras básicas para escrever os comandos SQL:**
 - Os comandos podem ser escritos em mais de uma linha;
 - Cláusulas diferentes são colocadas usualmente em linhas diferentes;
 - Podem ser usadas tabulações;
 - Comandos podem ser escritos em letras maiúsculas e/ou minúsculas;
 - Qualquer forma abaixo do comando é válida:

```
select * from aluno;
```

```
select  
*  
from aluno;
```

```
select *  
from aluno;
```

Escrevendo Comandos SQL

- **Um Bloco de Pesquisa Simples:**

- O comando select traz dados de uma tabela de banco de dados:
- Ex.: Para trazer todos os códigos, nomes e cidades de todos os alunos da tabela ALUNO:

```
SQL> select          cod_aluno, nom_aluno, nom_cidade  
      from  aluno;
```

- Note que o nome das colunas é separado por vírgulas.
- É possível também selecionar todas as colunas de uma tabela.
 - Ex.:

```
sql> select * from aluno;
```

- Outros itens que podem ser incluídos em uma cláusula select:

Escrevendo Comandos SQL

- **Expressões aritméticas em determinada coluna:**

```
SQL> select nom_curso, carga_horaria/4  
2 from curso;
```

NOM_CURSO	CARGA_HORARIA/4
PROGRAMAÇÃO DE COMPUTADORES	25
CÁLCULO NUMÉRICO	30
LINGUAGENS DE PROGRAMAÇÃO	25
BANCOS DE DADOS	50
ALGORITMOS E ESTRUTURAS DE DADOS	37.5
SISTEMAS OPERACIONAIS	

6 rows selected.

Considerando uma carga de 4 horas/dia o curso poderia ser dado em carga_horaria/4 dias

Escrevendo Comandos SQL

- **Apelidos (aliass) para colunas:**

```
SQL> select nom_curso, carga_horaria/4 dias  
2 from curso;
```

NOM_CURSO	DIAS
PROGRAMAÇÃO DE COMPUTADORES	25
CÁLCULO NUMÉRICO	30
LINGUAGENS DE PROGRAMAÇÃO	25
BANCOS DE DADOS	50
ALGORITMOS E ESTRUTURAS DE DADOS	37.5
SISTEMAS OPERACIONAIS	

```
6 rows selected.
```

Escrevendo Comandos SQL

- **Operadores de Concatenação:**

- Para concatenar duas colunas de uma tabela para um valor final específico:

```
SQL> select cod_curso||'-'||nom_curso  
2 from curso;
```

```
COD_CURSO||'-'||NOM_CURSO
```

```
-----  
1-PROGRAMAÇÃO DE COMPUTADORES  
2-CÁLCULO NUMÉRICO  
3-LINGUAGENS DE PROGRAMAÇÃO  
4-BANCOS DE DADOS  
5-ALGORITMOS E ESTRUTURAS DE DADOS  
6-SISTEMAS OPERACIONAIS
```

```
6 rows selected.
```

Escrevendo Comandos SQL

- **Tratamento de Valores Nulos:**

- Se determinada coluna não tem um valor, este valor é denominado **nulo**. Quando uma operação aritmética é feita com um valor nulo, o resultado é sempre nulo:

```
SQL> select nom_curso, carga_horaria/4 dias
      2  from curso;
```

NOM_CURSO	DIAS
PROGRAMAÇÃO DE COMPUTADORES	25
CÁLCULO NUMÉRICO	30
LINGUAGENS DE PROGRAMAÇÃO	25
BANCOS DE DADOS	50
ALGORITMOS E ESTRUTURAS DE DADOS	37.5
SISTEMAS OPERACIONAIS	

```
6 rows selected.
```

Escrevendo Comandos SQL

- Para a conversão de valores nulos (no exemplo para 100) utiliza-se a função NVL:

```
SQL> select nom_curso, nvl(carga_horaria,100)/4 dias  
2 from curso;
```

NOM_CURSO	DIAS
PROGRAMAÇÃO DE COMPUTADORES	25
CÁLCULO NUMÉRICO	30
LINGUAGENS DE PROGRAMAÇÃO	25
BANCOS DE DADOS	50
ALGORITMOS E ESTRUTURAS DE DADOS	37.5
SISTEMAS OPERACIONAIS	25

6 rows selected.

Escrevendo Comandos SQL

- **Evitando a seleção de valores idênticos em uma tabela:**
 - Na seleção de colunas de uma tabela sem nenhuma cláusula de distinção, são trazidos todos os valores mesmo que sejam idênticos:

```
SQL> select nom_cidade from aluno;
```

```
NOM_CIDADE
```

```
-----
```

```
BELO HORIZONTE
```

```
BELO HORIZONTE
```

```
MANAUS
```

```
NATAL
```

```
NATAL
```

```
CURITIBA
```

```
MANAUS
```

```
CAMPINAS
```

```
8 rows selected.
```

Escrevendo Comandos SQL

- **Utilizando a cláusula distinct:**

- Serve para selecionar valores distintos de determinadas colunas.

```
SQL> select distinct nom_cidade from aluno;
```

```
NOM_CIDADE
```

```
-----
```

```
BELO HORIZONTE
```

```
CAMPINAS
```

```
CURITIBA
```

```
MANAUS
```

```
NATAL
```

Escrevendo Comandos SQL

- **Ordenando colunas:**

- A cláusula `order by` é utilizada para trazer os dados em uma ordem específica.

```
SQL> select nom_aluno, nom_cidade from aluno  
2 order by nom_aluno;
```

NOM_ALUNO	NOM_CIDADE
ANA	NATAL
FELIPE	NATAL
JALENE	MANAUS
JOSÉ	BELO HORIZONTE
JOÃO	MANAUS
MARIA	BELO HORIZONTE
ROSA	CAMPINAS
TEREZA	CURITIBA

8 rows selected.

Escrevendo Comandos SQL

– Para mudar a ordem de pesquisa:

```
SQL> select nom_aluno, nom_cidade from aluno  
2 order by nom_aluno desc;
```

NOM_ALUNO	NOM_CIDADE
TEREZA	CURITIBA
ROSA	CAMPINAS
MARIA	BELO HORIZONTE
JOÃO	MANAUS
JOSÉ	BELO HORIZONTE
JALENE	MANAUS
FELIPE	NATAL
ANA	NATAL

8 rows selected.

Escrevendo Comandos SQL

- **A cláusula where:**

- A cláusula where corresponde ao operador relacional de restrição. Ela contém as condições que as linhas da relação devem satisfazer para serem mostradas.
- O where quando utilizado deve vir após a cláusula from.

```
SELECT      colunas
FROM        tabela
WHERE       condições que devem ser respeitadas.
```

- Operadores lógicos utilizados com a cláusula where:

Operador	Significado
=	igual a
>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a

Escrevendo Comandos SQL

- **Para listar os dados do curso com carga horária de 200hs:**

```
SQL> select cod_curso, nom_curso, carga_horaria
2   from   curso
3   where  carga_horaria=200;
```

```
COD_CURSO  NOM_CURSO
CARGA_HORARIA
```

```
-----  
-----
```

```
4 BANCOS DE DADOS
200
```

Escrevendo Comandos SQL

- **Para listar os cursos onde a carga horária é maior ou igual a 150:**

```
SQL> select nom_curso
      2  from   curso
      3  where  carga_horaria >= 150;
```

```
NOM_CURSO
```

```
-----  
CÁLCULO NUMÉRICO
```

```
BANCOS DE DADOS
```

```
ALGORITMOS E ESTRUTURAS DE DADOS
```

Escrevendo Comandos SQL

- **Comparando o valor entre duas colunas da tabela:**
 - Ex: Cursos onde carga horária atual é maior que a anterior

```
SQL> select nom_curso
      2  from   curso
      3  where  carga_horaria > carga_horaria_ant;
```

```
NOM_CURSO
```

```
-----  
CÁLCULO NUMÉRICO
```

```
BANCOS DE DADOS
```

```
ALGORITMOS E ESTRUTURAS DE DADOS
```

As colunas que estão sendo comparadas não precisam constar no resultado da query.

Escrevendo Comandos SQL

- **Operadores SQL:**

- Existem quatro operadores que podem ser utilizados em qualquer tipo de dados.

Operador	Significado:
between ___ and ____	valores que estão entre os dois valores especificados (inclusive os valores).
in (<i>lista</i>)	valor que seja igual a algum dos valores especificados na lista.
like	valores correspondentes ao valor especificado
is null	seleciona os valores nulos

Todos os operadores podem ser utilizados em suas formas negativas.

Escrevendo Comandos SQL

– Exemplos:

- Operador between:

```
SQL> select nom_curso, carga_horaria
2   from   curso
3   where  carga_horaria between 120 and 150;
```

NOM_CURSO	CARGA_HORARIA
CÁLCULO NUMÉRICO	120
ALGORITMOS E ESTRUTURAS DE DADOS	150

Na utilização do between o menor valor da comparação deve vir antes.

Escrevendo Comandos SQL

- Operador in

```
SQL> select nom_aluno  
2   from   aluno  
3   where  nom_cidade in ('MANAUS','NATAL');
```

```
NOM_ALUNO
```

```
-----
```

```
JOÃO
```

```
FELIPE
```

```
ANA
```

```
JALENE
```

Valores de caracteres devem vir entre aspas simples.

Escrevendo Comandos SQL

- Operador like

```
SQL> select nom_aluno from aluno where nom_aluno like 'A%';
```

```
NOM_ALUNO
```

```
-----
```

```
ANA
```

- O símbolo % significa qualquer caracter (ou conjunto) de caracteres.
- Outro operador que pode ser usado junto com a clausula like é o '_' (underscore) que substitui um número específico de caracteres.

```
SQL> select nom_aluno
```

```
2 from aluno
```

```
3 where nom_aluno like '____';
```

```
NOM_ALUNO
```

```
-----
```

```
JOSÉ
```

```
JOÃO
```

```
ROSA
```

A combinação dos dois operandos (% e _) também pode ser usada.

Escrevendo Comandos SQL

- Selecionar as cidades que contenham o substring 'NA' em qualquer posição.

```
SQL> select distinct nom_cidade
      2  from    aluno
      3  where   nom_cidade like '%NA%';
```

```
NOM_CIDADE
```

```
-----
```

```
CAMPINAS
```

```
MANAUS
```

```
NATAL
```

Escrevendo Comandos SQL

- Operador LIKE

```
SQL> select nom_cidade
      2  from    aluno
      3  where  nom_cidade like '__MP%';
```

```
NOM_CIDADE
```

```
-----
```

```
CAMPINAS
```

- Operador is null

```
SQL> select cod_curso, nom_curso, carga_horaria
      2  from    curso
      3  where  carga_horaria is null;
```

```
COD_CURSO  NOM_CURSO
```

```
CARGA_HORARIA
```

```
-----
```

```
6 SISTEMAS OPERACIONAIS
```

Escrevendo Comandos SQL

- **Selecionando dados com múltiplas condições.**
 - Os operadores and e or podem ser utilizados para compor expressões lógicas.
 - Exemplos:
 - Para encontrar todos os alunos de NATAL do sexo masculino:

```
SQL> select nom_aluno
      2  from    aluno
      3  where   nom_cidade = 'NATAL'
      4  and    sexo = 'M';
```

```
NOM_ALUNO
```

```
-----
```

```
FELIPE
```

Escrevendo Comandos SQL

- Para encontrar os alunos que são de NATAL ou do sexo masculino:

```
SQL> select nom_aluno
      2  from    aluno
      3  where   nom_cidade = 'NATAL'
      4  or     sexo = 'M';
```

```
NOM_ALUNO
```

```
-----
```

```
JOSÉ
```

```
JOÃO
```

```
FELIPE
```

```
ANA
```

Escrevendo Comandos SQL

- Podem ser utilizadas diversas combinações das cláusulas and/or:

```
SQL> select nom_aluno
      2  from    aluno
      3  where   sexo = 'F'
      4  and    nom_cidade = 'NATAL'
      5  or     nom_cidade = 'MANAUS';
```

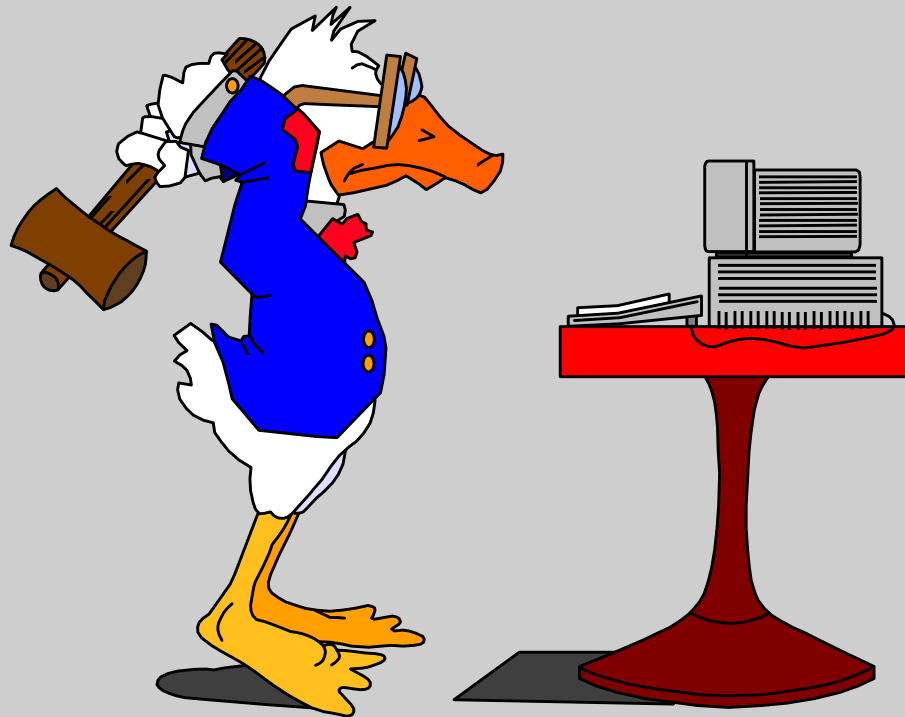
```
NOM_ALUNO
```

```
-----
```

```
JOÃO
```

```
ANA
```

```
JALENE
```



Variáveis de Substituição em Queries

Variáveis de Substituição em Queries

- Uma **Query** que está mantida em um buffer ou em um arquivo .sql pode conter variáveis que serão informadas na execução da query:

```
SQL> select cod_aluno, nom_aluno, sexo
      2  from  aluno
      3  where nom_cidade = &cidade;
```

```
Enter value for cidade: 'NATAL'
```

```
old  3: where  nom_cidade = &cidade
```

```
new  3: where  nom_cidade = 'NATAL'
```

```
COD_ALUNO  NOM_ALUNO                S
-----  -
          4 FELIPE                    M
          5 ANA                        F
```

Variáveis de Substituição em Queries

- Com o símbolo **&&** a variável é pedida apenas a primeira vez. Nas outras vezes ela já assume o valor previamente especificado.

```
SQL> select cod_aluno, nom_aluno, sexo
  2  from  aluno
  3  where nom_cidade = &&cidade;
```

Enter value for cidade: 'NATAL'

old 3: where nom_cidade = &&cidade

new 3: where nom_cidade = 'NATAL'

COD_ALUNO	NOM_ALUNO	SEXO
4	FELIPE	M
5	ANA	F

Para substituir o valor de uma variável ou criar uma nova variável && usa-se o comando define.

Variáveis de Substituição em Queries

- **Comando Define:**

```
SQL> define dif = carga_horaria-carga_horaria_ant
```

```
SQL> select cod_curso, &dif
```

```
  2 from curso;
```

```
old  1: select cod_curso, &dif
```

```
new  1: select cod_curso, carga_horaria-carga_horaria_ant
```

```
COD_CURSO  CARGA_HORARIA-CARGA_HORARIA_ANT
```

```
-----
```

1	0
2	20
3	-20
4	50
5	30
6	

```
6 rows selected.
```

Variáveis de Substituição em Queries

- **Comando Accept:**

- O comando accept permite uma variável ser criada e um valor que vai ser entrado na execução do SQL ser armazenado. O accept também pode ser usado em um arquivo .sql.
- Com o comando accept o usuário pode:
 - checar o tipo de dado que está sendo informado (numérico ou caracter);
 - incluir texto explicativo sobre a variável que está sendo pedida;
 - A entrada de dados pode ser invisível (para casos da entrada de passwords).

- **OBS**

- O comando ACCEPT não deve ser copiado para a área de transferência seguido de outros comandos e colado no SQL*Plus, pois este interpretará os comandos posteriores como a entrada do ACCEPT.

Variáveis de Substituição em Queries

– Sintaxe:

```
ACC[EPT] variável [NUMBER / CHAR ] [ PROMPT / NOPROMPT 'texto' ] [ HIDE  
]
```

- Ex.:

```
SQL> accept cidade char prompt 'Informe a cidade: ' ;  
Informe a cidade: NATAL  
SQL> accept password char prompt 'Informe a senha: ' hide;  
Informe a senha: *****  
SQL> accept cidade char noprompt;  
CAMPINAS  
SQL>
```

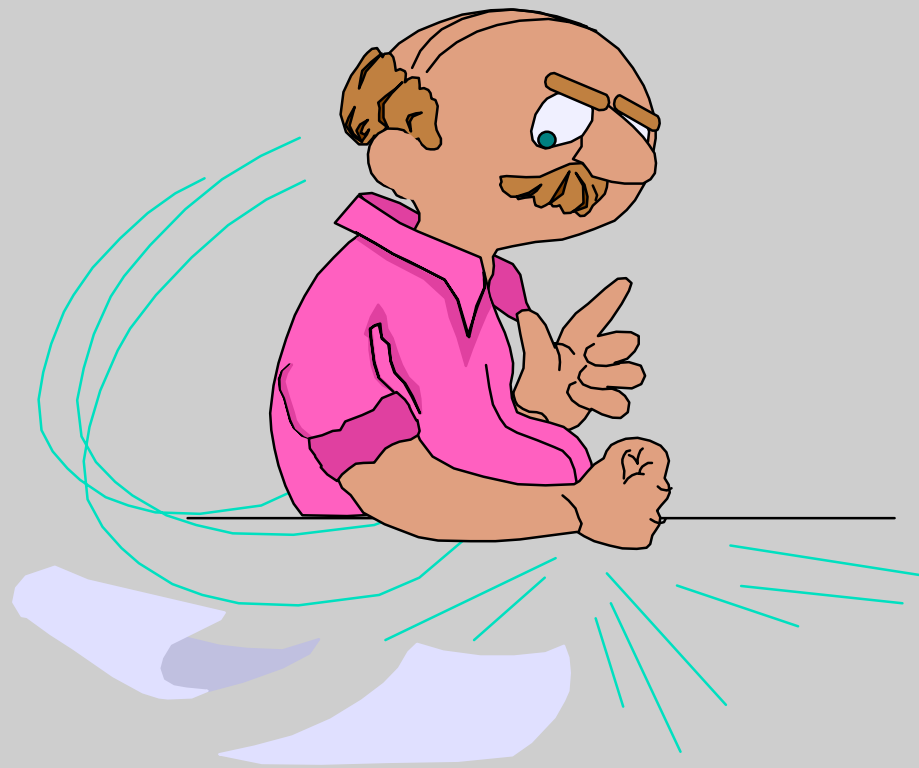
```
SQL> define  
DEFINE NOTA                =          100 (NUMBER)  
DEFINE PASSWORD            =          "senha" (CHAR)  
DEFINE NOTA2               =          75 (NUMBER)  
SQL>
```

Módulo III

- ⇒ Funções
- ⇒ Extração de Dados de mais de uma Tabela
- ⇒ Subqueries
- ⇒ Linguagem de Manipulação de Dados (DML)

Introdução ao

Oracle7



Funções

Funções

- **Objetivos desta unidade:**

- Introduzir os diversos tipos de funções: funções de datas, de conversão, etc.

- **Introdução às funções:**

- **Funções** são usadas para manipulações de dados. Elas aceitam um ou mais argumentos e retornam um valor. O formato de uma função é geralmente o seguinte:

nome_da_função (argumento_1, argumento_2.....)

- **Funções** podem ser usadas para:

- Fazer cálculos em dados;
- modificar dados;
- manipular a saída de grupos de linhas;
- alterar o formato de datas;
- converter tipos de dados de colunas.

Funções

- Os tipos de funções existentes são:
 - Caracter;
 - Números;
 - Data
 - Conversão;
 - Grupo
- Algumas funções têm efeito em apenas uma linha, outras têm efeito em um grupo de linhas.

Funções de Caracter

- **Funções de Caracter.**

- Função **Lower**:

- Recupera o dado especificado em letra minúscula:

```
SQL> select lower(nom_cidade) from aluno;
```

```
LOWER(NOM_CIDADE)
```

```
-----
```

```
belo horizonte
```

```
belo horizonte
```

```
manaus
```

```
natal
```

```
natal
```

```
curitiba
```

```
manaus
```

```
campinas
```

```
8 rows selected.
```

Funções de Caracter

– Função Upper:

- Recupera o dado especificado em letra maiúscula:

```
SQL> select upper(nom_cidade) from aluno;
```

```
UPPER(NOM_CIDADE)
```

```
-----
```

```
BELO HORIZONTE
```

```
BELO HORIZONTE
```

```
MANAUS
```

```
NATAL
```

```
NATAL
```

```
CURITIBA
```

```
MANAUS
```

```
CAMPINAS
```

```
8 rows selected.
```

Funções de Caracter

– Função Initcap:

- Recupera o dado especificado com a primeira letra maiúscula:

```
SQL> select initcap(nom_cidade) from aluno;
```

```
INITCAP (NOM_CIDADE)
```

```
-----
```

```
Belo Horizonte
```

```
Belo Horizonte
```

```
Manaus
```

```
Natal
```

```
Natal
```

```
Curitiba
```

```
Manaus
```

```
Campinas
```

```
8 rows selected.
```

Funções de Caracter

– Funções LPAD e RPAD:

- A função LPAD completa com o caracter(es) informado, a esquerda da coluna, a coluna/valor até o número especificado em n.

– LPAD (col|val,n,'caracter')

Ex.:

```
SQL> select lpad(nom_cidade, 30, '*') from aluno;
```

```
LPAD (NOM_CIDADE,30, '*')
```

```
-----
*****BELO HORIZONTE
*****BELO HORIZONTE
*****MANAUS
*****NATAL
*****NATAL
*****CURITIBA
*****MANAUS
*****CAMPINAS
```

```
8 rows selected.
```

Funções de Caracter

- A função **RPAD** completa com o(s) caracter(es) informado, a direita da coluna, a coluna/valor até o número especificado em n.

– Ex.:

```
SQL> select rpad(nom_cidade, 30, '*') from aluno;
```

```
RPAD(NOM_CIDADE,30,'*')
```

```
-----  
BELO HORIZONTE*****  
BELO HORIZONTE*****  
MANAUS*****  
NATAL*****  
NATAL*****  
CURITIBA*****  
MANAUS*****  
CAMPINAS*****
```

```
8 rows selected.
```

Funções de Caracter

– Função **Substr**:

- Traz uma parte do dado especificado:

SUBSTR (col|val,pos,n)

```
SQL> select substr('SQL*PLUS', 1, 3), substr(nom_cidade, 3),
2  substr(nom_cidade, 4,5)
3  from  aluno;
```

```
SUB SUBSTR(NOM_CIDADE, SUBST
```

```
--- -----
SQL LO HORIZONTE      O HOR
SQL LO HORIZONTE      O HOR
SQL NAUS              AUS
SQL TAL               AL
SQL TAL               AL
SQL RITIBA            ITIBA
SQL NAUS              AUS
SQL MPINAS            PINAS
```

```
8 rows selected.
```

Funções de Caracter

– Funções LTRIM e RTRIM

- São utilizadas para remover caracteres especificados de uma coluna/valor especificado:

LTRIM (col|val,'caracter')

- Remove da esquerda da coluna|valor a(s) ocorrência(s) encontradas do caracter informado ou da combinação deles quando informado mais de um caracter.

Funções de Caracter

- Exemplo:

```
SQL> select nom_cidade, ltrim(nom_cidade, 'B'), ltrim(nom_cidade, 'BNA')
       2 from aluno;
```

NOM_CIDADE	LTRIM(NOM_CIDADE, 'B')	LTRIM(NOM_CIDADE, 'BN
-----	-----	-----
BELO HORIZONTE	ELO HORIZONTE	ELO HORIZONTE
BELO HORIZONTE	ELO HORIZONTE	ELO HORIZONTE
MANAUS	MANAUS	MANAUS
NATAL	NATAL	TAL
NATAL	NATAL	TAL
CURITIBA	CURITIBA	CURITIBA
MANAUS	MANAUS	MANAUS
CAMPINAS	CAMPINAS	CAMPINAS

8 rows selected.

Funções de Caracter

– Rtrim (col|val,'character')

- Remove da direita da coluna|valor a(s) ocorrência(s) encontradas do caracter informado ou da combinação deles quando informado mais de um caracter.

```
SQL> select nom_cidade, rtrim(nom_cidade, 'S'), rtrim(nom_cidade, 'AUS')
       2 from aluno;
```

NOM_CIDADE	RTRIM(NOM_CIDADE, 'S')	RTRIM(NOM_CIDADE, 'AUS')
-----	-----	-----
BELO HORIZONTE	BELO HORIZONTE	BELO HORIZONTE
BELO HORIZONTE	BELO HORIZONTE	BELO HORIZONTE
MANAUS	MANAU	MAN
NATAL	NATAL	NATAL
NATAL	NATAL	NATAL
CURITIBA	CURITIBA	CURITIB
MANAUS	MANAU	MAN
CAMPINAS	CAMPINA	CAMPIN

8 rows selected.

Funções de Caracter

– Função Length:

- Retorna o número de caracteres de uma coluna/valor especificado:

LENGTH (col|val)

– Ex.:

```
SQL> select length('ORACLE'), length(nom_aluno), length(nom_cidade)
       2 from aluno;
```

LENGTH('ORACLE')	LENGTH(NOM_ALUNO)	LENGTH(NOM_CIDADE)
6	4	14
6	5	14
6	4	6
6	6	5
6	3	5
6	6	8
6	6	6
6	4	8

8 rows selected.

Funções Numéricas

- **Funções numéricas.**

- As **funções numéricas** recebem como argumentos dados numéricos e retornam dados numéricos.

- **Round** (col|value,n)

- Arredonda o valor / coluna especificados com o número de casas decimais informado:

```
SQL> select round(45.923, 1), round(45.963, 1) from dual;
```

```
ROUND(45.923,1) ROUND(45.963,1)
```

```
-----
```

```
45.9
```

```
46
```

Funções Numéricas

- **Power** (col|value,n)

- Função de exponenciação. Eleva o número/coluna especificados por n.

```
SQL> select power(carga_horaria, 0.5) from curso;
```

```
POWER(CARGA_HORARIA, 0.5)
```

```
-----
```

```
10
```

```
10.954451
```

```
10
```

```
14.142136
```

```
12.247449
```

```
6 rows selected.
```

Funções Numéricas

– **SQRT** (col|value)

- Função de raiz quadrada. Faz o cálculo da raiz quadrada do número/coluna especificado.

```
SQL> select sqrt(carga_horaria) from curso;
```

```
SQRT(CARGA_HORARIA)
```

```
-----
```

```
          10  
    10.954451  
          10  
    14.142136  
    12.247449
```

```
6 rows selected.
```

Funções de Datas

- **Funções de datas** operam com datas no Oracle. Todas as funções exceto **Months Between**, retornam um valor no formato de data.
- O Oracle guarda datas com os seguintes dados:
 - Século
 - Ano
 - Mês
 - Dia
 - Hora
 - Minutos
 - Segundos
- O formato default da data no Oracle é: **dd-mon-yy** e armazena datas que podem ir de 1 de janeiro de 4712 antes de Cristo até o ano 4712 depois de Cristo.

Funções de Datas

– Coluna Sysdate.

- A coluna Sysdate é uma pseudo-coluna que retorna o valor de data e hora. Você pode usar o Sysdate como qualquer outro tipo de coluna. Ela é usualmente selecionada da tabela *dummy* dual.
- Para mostrar a data corrente use o seguinte comando:

```
SQL> select sysdate from sys.dual;
```

```
SYSDATE
```

```
-----
```

```
21-NOV-95
```

– Usando operadores aritméticos com campo data:

- Com o campo data do Oracle é possível fazer cálculos usando operadores aritméticos.
- É possível adicionar números a datas e fazer cálculos com duas datas.

Funções de Datas

- Operações que podem ser feitas com datas:

data + número	adiciona um número de dias à data produzindo uma nova data
data - número	subtrai um número de dias da data produzindo uma nova data
data - data	subtrai uma data de outra produzindo um número de dias.
data + número / 24	adiciona um número de horas na data produzindo uma nova data.

```
SQL> select dat_inicial, dat_inicial-7, dat_inicial+7
       2 ,sysdate-dat_inicial from periodo_letivo;
```

```
DAT_INICI  DAT_INICI  DAT_INICI  SYSDATE-DAT_INICIAL
-----  -----  -----  -----
01-MAR-94  22-FEB-94  08-MAR-94           1042.4246
01-AUG-94  25-JUL-94  08-AUG-94           889.42456
01-MAR-95  22-FEB-95  08-MAR-95           677.42456
01-AUG-95  25-JUL-95  08-AUG-95           524.42456
```

Funções de Datas

– Função Months Between:

- A função Months Between encontra o número de meses entre as duas datas informadas. O resultado pode ser positivo ou negativo.

```
SQL> select months_between(sysdate, dat_inicial),
2         months_between('01-jan-97', '01-jan-96')
3 from   periodo_letivo;
```

MONTHS_BETWEEN(SYSDATE, DAT_INICIAL)	MONTHS_BETWEEN('01-JAN-97', '01-JAN-96')
34.175013	12
29.175013	12
22.175013	12
17.175013	12

Funções de Conversão

- O **SQL** possui funções para conversão de tipos de dados. Essas funções convertem um tipo de dado para outro.
 - Função **To Char** :
 - É utilizada geralmente para mudar um formato de uma data para outro.

```
SQL> select to_char(sysdate, 'DD/MM/YYYY')  
2 from sys.dual;
```

```
TO_CHAR(SYSDATE, 'DD/MM/YYYY')
```

```
-----
```

```
06/01/1997
```

Funções de Conversão

– Para mostrar uma data/hora:

```
SQL> select to_char(sysdate, 'DD/MM/YYYY HH24:MI')  
       2 from sys.dual;
```

```
TO_CHAR(SYSDATE, 'DD/MM/YYYYHH24:MI')
```

```
-----
```

```
06/01/1997 10:14
```

– Para mostrar apenas uma hora:

```
SQL> select to_char(sysdate, 'HH24:MI:SS')  
       2 from sys.dual;
```

```
TO_CHAR(SYSDATE, 'HH24:MI:SS')
```

```
-----
```

```
10:14:52
```

Funções de Conversão

– Função To Number:

- É usada para transformar um número que está no formato de caracter para o formato numérico:

```
SQL> select nom_curso, carga_horaria from curso
      2  where carga_horaria > to_number('100');
```

NOM_CURSO	CARGA_HORARIA
CÁLCULO NUMÉRICO	120
BANCOS DE DADOS	200
ALGORITMOS E ESTRUTURAS DE DADOS	150

Funções de Conversão

– Função To Date:

- Transforma um conjunto de caracteres em uma data válida.

```
SQL> select *  
  2  from  periodo_letivo  
  3  where dat_inicial = to_date('01/03/1995', 'DD/MM/YYYY');
```

```
COD_PERIODO  DAT_INICI  DAT_FINAL  
-----  
          3  01-MAR-95  30-JUN-95
```

Funções de Conversão

– Função Decode.

- A função Decode facilita queries condicionais, fazendo o trabalho das funções Case ou IF-THEN-ELSE.
- Sintaxe:

`decode (col|expression , search1,result1[search2,result2] default)`

```
SQL> select nom_aluno, decode(sexo, 'M', 'MASCULINO', 'F','FEMENINO') gênero
      2 from aluno;
```

NOM_ALUNO	GÊNERO
JOSÉ	MASCULINO
MARIA	FEMENINO
JOÃO	MASCULINO
FELIPE	MASCULINO
ANA	FEMENINO
TEREZA	FEMENINO
JALENE	FEMENINO
ROSA	FEMENINO

8 rows selected.

Funções de Grupo

- As **funções de grupo** agem sempre sobre um conjunto de linhas ao invés de agir sobre cada linha separada.

- Função **Avg**:

- Calcula a média dos valores selecionados:

```
SQL> select avg(carga_horaria)
      2  from curso;
```

```
AVG(CARGA_HORARIA)
```

```
-----
```

```
134
```

Funções de Grupo

– Função Min / Max:

- Seleciona o menor ou o maior valor dos valores selecionados:

```
SQL> select min(carga_horaria)
2   from   curso;
```

```
MIN(CARGA_HORARIA)
-----
100
```

```
SQL> select count(*)
2   from   curso
3  where  carga_horaria <> carga_horaria_ant;
```

```
COUNT(*)
-----
4
```

Funções de Grupo

– Função **Sum**:

- A função **Sum** faz o somatório dos dados extraídos da coluna especificada ignorando os valores **Null**:

```
SQL> select sum(carga_horaria)
      2  from  curso;
```

```
SUM(CARGA_HORARIA)
-----
                    670
```

– Função **Count**:

- A função **Count** faz a contagem das linhas retornadas em uma Query:

```
SQL> select count(carga_horaria) from curso;
```

```
COUNT(CARGA_HORARIA)
-----
                    5
```

Funções de Grupo

– Cláusula Group BY:

- A Cláusula group by é utilizada para dividir a tabela em conjuntos menores de dados relacionados entre si.
- Ex.:
- Para contar os alunos por cidade:

```
SQL> select nom_cidade, count(*)  
2   from   aluno  
3   group by nom_cidade;
```

NOM_CIDADE	COUNT (*)
-----	-----
BELO HORIZONTE	2
CAMPINAS	1
CURITIBA	1
MANAUS	2
NATAL	2

Funções de Grupo

- O **where** pode ser usado em conjunto com a cláusula group by para excluir determinada linha:

```
SQL> select nom_cidade, count(*)
2   from   aluno
3   where  sexo = 'M'
4   group by nom_cidade;
```

NOM_CIDADE	COUNT (*)
-----	-----
BELO HORIZONTE	1
MANAUS	1
NATAL	1

Funções de Grupo

– Cláusula Having:

- A cláusula Having é utilizada para selecionar os grupos retornados de uma seleção feita com Group By .
- Ex.:

Para selecionar as cidades com mais de dois alunos:

```
SQL> select nom_cidade
2   from   aluno
3   group by nom_cidade
4   having count(*) >=2;
```

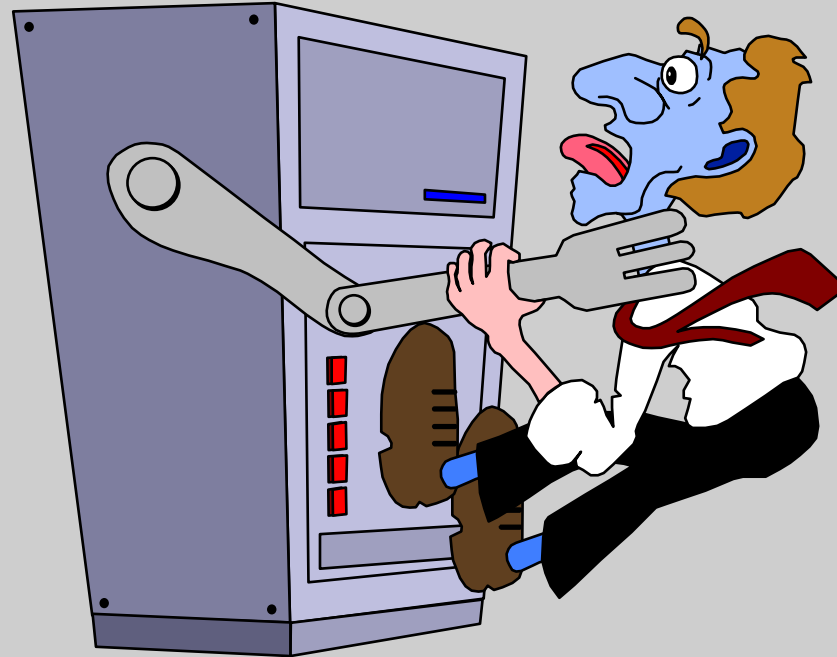
```
NOM_CIDADE
```

```
-----
```

```
BELO HORIZONTE
```

```
MANAUS
```

```
NATAL
```



Extração de Dados de mais de uma Tabela

Joins

- Em um **banco de dados relacional**, é comum fazer uma seleção em várias tabelas relacionadas para obtermos um resultado desejado.
- Um **Join** é usado quando uma Query necessita de dados de mais de uma tabela do database, o **Join** é feito comparando colunas das tabelas envolvidas na relação.
- Equi-join
 - **Equi-join** é usado quando o valor da coluna de uma tabela tem o seu valor correspondente na segunda.
 - Nas tabelas ALUNO e MATRICULA os valores comparados são da coluna cod_aluno que tem em ambas as tabelas e tem o mesmo valor. No equi-join o operador '=' é usado.
 - Ex.: Para sabermos o nome do aluno de uma matrícula, nós devemos comparar os valores da coluna cod_aluno das duas tabelas desta relação. As colunas são trazidas das duas tabelas envolvidas.

Joins

- Selecionar o nome dos alunos que se matricularam no período letivo 4:

```
SQL> select distinct nom_aluno
  2  from   matricula, aluno
  3  where  matricula.cod_aluno = aluno.cod_aluno
  4  and    cod_perodo = 4;
```

```
NOM_ALUNO
```

```
-----
```

```
ANA
```

```
FELIPE
```

```
JALENE
```

```
ROSA
```

```
TEREZA
```

Joins

- Para especificar de qual tabela queremos que o valor seja trazido usamos o seguinte formato:

```
SQL> select distinct matricula.cod_aluno, nom_aluno
  2   from   matricula, aluno
  3   where  matricula.cod_aluno = aluno.cod_aluno
  4   and    cod_perodo = 4
  5   order by nom_aluno;
```

```
COD_ALUNO  NOM_ALUNO
```

```
-----
  5 ANA
  4 FELIPE
  7 JALENE
  8 ROSA
  6 TEREZA
```

Usando Apelidos para as Tabelas (Table Aliases)

- Para simplificar o uso do nome das tabelas nas Queries, podem ser criados apelidos para as tabelas para serem usados na Query.

```
SQL> select distinct nom_aluno
  2  from    matricula m, aluno a
  3  where   m.cod_aluno = a.cod_aluno
  4  and     cod_periodo = 4;
```

```
NOM_ALUNO
```

```
-----
```

```
ANA
```

```
FELIPE
```

```
JALENE
```

```
ROSA
```

```
TEREZA
```

Usando Apelidos para as Tabelas (Table Aliases)

- Join usando a mesma tabela:
- Usando os apelidos de uma tabela (table aliases) é possível fazer um join de uma tabela com ela mesma.
- Ex.: Pares de alunos da mesma cidade

```
SQL> select a1.nom_aluno, a2.nom_aluno
2  from   aluno a1, aluno a2
3  where  a1.nom_cidade = a2.nom_cidade;
```

Usando Apelidos para as Tabelas (Table Aliases)

```
SQL> select a1.nom_aluno, a2.nom_aluno  
2   from   aluno a1, aluno a2  
3   where  a1.nom_cidade = a2.nom_cidade;
```

NOM_ALUNO	NOM_ALUNO
-----	-----
JOSÉ	JOSÉ
MARIA	JOSÉ
JOSÉ	MARIA
MARIA	MARIA
ROSA	ROSA
TEREZA	TEREZA
JOÃO	JOÃO
JALENE	JOÃO
JOÃO	JALENE
JALENE	JALENE
FELIPE	FELIPE
ANA	FELIPE
FELIPE	ANA
ANA	ANA

14 rows selected.

Usando Apelidos para as Tabelas (Table Aliases)

- Para remover repetições e redundâncias do tipo (x,y) (y,x):

```
SQL> select a1.nom_aluno, a2.nom_aluno
  2  from   aluno a1, aluno a2
  3  where  a1.nom_cidade = a2.nom_cidade
  4  and    a1.cod_aluno < a2.cod_aluno;
```

NOM_ALUNO

NOM_ALUNO

JOSÉ

MARIA

FELIPE

ANA

JOÃO

JALENE

Outros Tipos de Join

- **União, interseção e subtração**, (union, intersect e minus) são comandos para construir queries que se referem a mais de uma tabela. Estes operadores combinam mais de um select para obter um resultado desejado.

- **Union**:

- **União**: usado para selecionar todas as linhas distintas retornadas de duas queries:

```
SQL> select nom_cidade
      2  from   aluno
      3  where  ano_ingresso=1992
      4  union
      5  select nom_cidade
      6  from   aluno
      7  where  ano_ingresso=1993;
```

```
NOM_CIDADE
```

```
-----
```

```
BELO HORIZONTE
```

```
CAMPINAS
```

```
CURITIBA
```

```
NATAL
```

Outros Tipos de Join

– Intersect:

- Interseção: retorna apenas as linhas que foram selecionadas nos dois selects.

-- Intersecao das cidades de alunos que ingressaram em 1992 e 1993

```
SQL> select nom_cidade
2   from   aluno
3   where  ano_ingresso=1992
4   intersect
5   select nom_cidade
6   from   aluno
7   where  ano_ingresso=1993;
```

```
NOM_CIDADE
```

```
-----
```

```
BELO HORIZONTE
```

Outros Tipos de Join

- **Subtração**: Usado para trazer as linhas retornadas pela primeira Query que não estão na segunda:

```
-- Cidades onde há alunos que ingressaram em 1992 mas não em 1993
```

```
SQL> select nom_cidade
  2  from    aluno
  3  where   ano_ingresso=1992
  4  minus
  5  select  nom_cidade
  6  from    aluno
  7  where   ano_ingresso=1993;
```

```
NOM_CIDADE
```

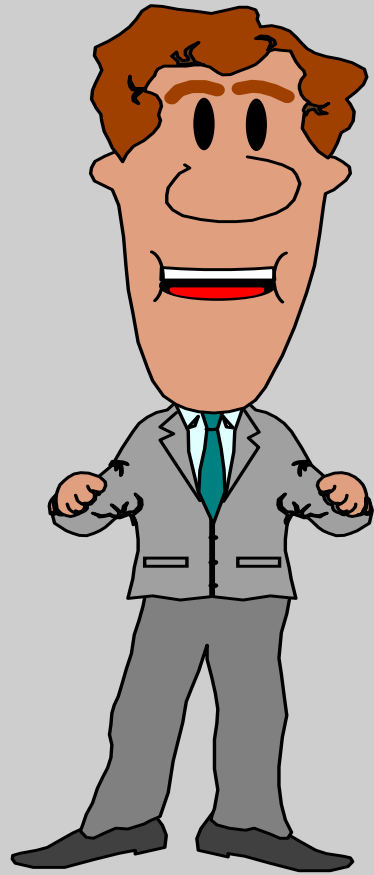
```
-----
```

```
CURITIBA
```

Outros Tipos de Join

É possível construir **Queries** com diversos operadores, a ordem de execução é da primeira Query para a última.

- Regras para o uso dos operadores Union, Intersect e Minus:
 - Os comandos selects das queries devem ter o mesmo número de colunas;
 - As colunas correspondentes devem ser do mesmo tipo de dados;
 - Linhas duplicadas são automaticamente eliminadas (não pode ser usada a cláusula Distinct);
 - O nome das colunas da primeira Query serão usadas no resultado;
 - Order by só pode ser usado no final do comando;
 - Os comandos select são executados do primeiro para o último.



Subqueries

Subqueries

- Uma **Subquery** é um comando select dentro de outro comando select .

– Ex.:

```
SQL> select column1, column2, ...  
      from table  
      where column=  
              (select column  
               from table  
               where condition)
```

- Este tipo de estrutura é geralmente utilizada quando precisamos selecionar linhas de uma tabela com uma condição que depende dos dados que estão na própria tabela.
- Para encontrarmos o curso com a menor carga horária, sendo que não sabemos qual é a menor carga horária (este dado também está na tabela e pode variar com o tempo).

Subqueries

- Primeiro vamos encontrar a menor carga horária:

```
select min(carga_horaria) from curso;
```

```
MIN(CARGA_HORARIA)
-----
                100
```

- A seguir devemos selecionar o curso:

```
select nom_curso, carga_horaria
from   curso
where  carga_horaria = 100;
```

- ou fazer os dois selects juntos:

```
SQL> select nom_curso, carga_horaria
      2  from   curso
      3  where  carga_horaria = (select min(carga_horaria) from curso);
```

```
NOM_CURSO                                CARGA_HORARIA
-----
PROGRAMAÇÃO DE COMPUTADORES                100
LINGUAGENS DE PROGRAMAÇÃO                  100
```

Subqueries

- Neste caso a Query mais interna é executada primeiro e traz como resultado o valor 100. A seguir a Query principal é executada substituindo a Query interna pelo valor encontrado.
- No exemplo dado a Query interna volta um único valor. Neste caso um comparador lógico deve ser usado, por exemplo: “ = , < , > , >= , etc. “
- Ex.: Alunos que ingressaram no mesmo ano da aluna MARIA

```
SQL> select nom_aluno, ano_ingresso
  2   from   aluno
  3   where  ano_ingresso = (select ano_ingresso
  4                               from   aluno
  5                               where  nom_aluno = 'MARIA' );
```

NOM_ALUNO	ANO_INGRESSO
MARIA	1993
FELIPE	1993
ROSA	1993

Comparando mais de um Valor:

- A **Query** seguinte tenta encontrar, para cada cidade, os alunos que ingressaram primeiro

```
SQL> select nom_aluno, nom_cidade, ano_ingresso
  2  from aluno
  3  where ano_ingresso in (select min(ano_ingresso)
  4                          from aluno
  5                          group by nom_cidade);
```

NOM_ALUNO	NOM_CIDADE	ANO_INGRESSO
JALENE	MANAUS	1991
JOSÉ	BELO HORIZONTE	1992
TEREZA	CURITIBA	1992
MARIA	BELO HORIZONTE	1993
ROSA	CAMPINAS	1993
FELIPE	NATAL	1993

6 rows selected.

Comparando mais de um Valor

- Neste caso a Query interna contém uma cláusula group by isto significa que ela pode retornar mais de um valor.
- A Query interna retorna simplesmente os menores anos de ingresso das cidades. Se houvesse um aluno de uma cidade que tivesse ano de ingresso igual ao menor ano de ingresso de outra cidade ele também seria selecionado. Neste caso a Query não traria o resultado desejado.
- A seguir mostramos duas versões da query correta.

Comparando mais de um Valor

- Versão correta 1

```
SQL> select nom_aluno, nom_cidade, ano_ingresso
2  from aluno
3  where (nom_cidade, ano_ingresso) in
4         ( select nom_cidade, min(ano_ingresso)
5           from aluno
6           group by nom_cidade);
```

NOM_ALUNO	NOM_CIDADE	ANO_INGRESSO
JOSÉ	BELO HORIZONTE	1992
ROSA	CAMPINAS	1993
TEREZA	CURITIBA	1992
JALENE	MANAUS	1991
FELIPE	NATAL	1993

Comparando mais de um Valor

- Versão correta 2

```
SQL> select nom_aluno, nom_cidade, ano_ingresso
  2  from    aluno a
  3  where   ano_ingresso in (select min(ano_ingresso)
  4                               from    aluno
  5                               where   nom_cidade=a.nom_cidade
  6                               group   by nom_cidade);
```

NOM_ALUNO	NOM_CIDADE	ANO_INGRESSO
JOSÉ	BELO HORIZONTE	1992
FELIPE	NATAL	1993
TEREZA	CURITIBA	1992
JALENE	MANAUS	1991
ROSA	CAMPINAS	1993

Tipos de Erros Encontrados

- Quando uma **SubQuery** retorna mais de um valor e um comparador simples é usado, o SQL retorna a seguinte mensagem:

```
SQL> select nom_aluno, nom_cidade, ano_ingresso
2  from    aluno
3  where   ano_ingresso = (select min(ano_ingresso)
4                          from    aluno
5                          group by nom_cidade);
```

ERROR:

```
ORA-01427: single-row subquery returns more than one row
no rows selected
```

Operadores ANY e ALL

- Os operadores Any e All são usados em subqueries que retornam mais de uma linha. Eles são usados na cláusula where em conjunto com os operadores lógicos: (= , != , > , < , >= , <=).
- O Operador Any compara um valor com cada valor retornado pela Query.
- Ex.:
 - Para mostrar os alunos que ingressaram após algum aluno de Belo Horizonte

Operadores ANY e ALL

```
SQL> select nom_aluno, nom_cidade, ano_ingresso
2  from aluno
3  where ano_ingresso > any (select ano_ingresso
4                             from aluno
5                             where nom_cidade = 'BELO HORIZONTE');
```

NOM_ALUNO	NOM_CIDADE	ANO_INGRESSO
MARIA	BELO HORIZONTE	1993
JOÃO	MANAUS	1994
FELIPE	NATAL	1993
ANA	NATAL	1995
ROSA	CAMPINAS	1993

Operadores ANY e ALL

- O operador **ALL** compara um valor com todos os valores retornados pela Subquery.
- Alunos que ingressaram após todos os alunos de Belo Horizonte

```
SQL> select nom_aluno, nom_cidade, ano_ingresso
2  from aluno
3  where ano_ingresso > all (select ano_ingresso
4                             from aluno
5                             where nom_cidade = 'BELO HORIZONTE');
```

NOM_ALUNO	NOM_CIDADE	ANO_INGRESSO
JOÃO	MANAUS	1994
ANA	NATAL	1995

O **maior ano de ingresso** de Belo Horizonte é 1993, portanto a Query vai retornar os anos de ingresso maiores que 1993. O **Not** pode ser usado com os operadores IN, ANY ou ALL.

Cláusula Having

– Cláusula Having:

- A cláusula having é utilizada para selecionar linhas de uma seleção feita com group by . A cláusula having é muito utilizada em subqueries.
- Ex.: Para as cidades com mais de um aluno, retornar o menor ano de ingresso

```
SQL> select nom_cidade, min(ano_ingresso)
2   from   aluno
3   group by nom_cidade
4   having count(*) > 1;
```

NOM_CIDADE	MIN(ANO_INGRESSO)
-----	-----
BELO HORIZONTE	1992
MANAUS	1991
NATAL	1993

Cláusula Having

- Ex.: Alunos cuja média geral é superior à média da aluna MARIA (cod_aluno = 2)

```
SQL> select nom_aluno, avg(val_nota_final)
2   from   matricula m, aluno a
3   where  m.cod_aluno = a.cod_aluno
4   group by nom_aluno
5   having avg(val_nota_final) > (select avg(val_nota_final)
6                                   from   matricula
7                                   where  cod_aluno=2);
```

NOM_ALUNO	AVG (VAL_NOTA_FINAL)
ANA	78
FELIPE	70.714286
JALENE	71.666667
JOÃO	82.5
ROSA	93.8
TEREZA	88

6 rows selected.

Cláusula Having

- Se não soubéssemos o código da aluna MARIA poderíamos escrever:

```
SQL> select nom_aluno, avg(val_nota_final)
  2  from   matricula m, aluno a
  3  where  m.cod_aluno = a.cod_aluno
  4  group by nom_aluno
  5  having avg(val_nota_final) >
  6         ( select avg(val_nota_final)
  7           from   matricula
  8           where  cod_aluno =
  9                 ( select cod_aluno
 10                   from   aluno
 11                   where  nom_aluno = 'MARIA' ) );
```

Cláusula Having

- Aluno com maior média de notas:

```
SQL> select nom_aluno, avg(val_nota_final)
  2  from   matricula m, aluno a
  3  where  m.cod_aluno = a.cod_aluno
  4  group  by nom_aluno
  5  having avg(val_nota_final) = (select max(avg(val_nota_final))
  6                                     from   matricula
  7                                     group  by cod_aluno);
```

NOM_ALUNO	AVG (VAL_NOTA_FINAL)
ROSA	93.8

Uma **Subquery** não pode conter a cláusula order by. Não existem limites de subqueries aninhadas.

Subqueries: Sumário

- A Query interna deve ser usada dentro de parênteses;
- A SubQuery não pode conter a cláusula order by;
- Quando utilizando múltiplas colunas em uma Subquery, estas colunas devem aparecer na mesma ordem e com os mesmos tipos de dados da Query principal, além do mesmo número de colunas;
- Podem ser usados operadores lógicos bem como os operadores Any e All.
- As cláusulas numa instrução SQL são executadas na seguinte ordem:
 - from (produto cartesiano)
 - where (restrição)
 - group by (agrupamento das linhas que satisfizeram WHERE)
 - having (restrição sobre linhas do agrupamento)



Linguagem de Manipulação de Dados (DML)

Inserindo Novas Linhas na Tabela

Inserindo novas linhas na tabela:

- O comando insert é usado para adicionar linhas em uma tabela.

- Sintaxe:

```
insert into tabela [ ( coluna,coluna... ) ]  
    values ( valor,valor,...);
```

- Para inserir um novo periodo na tabela PERIODOS_LETIVOS:

```
insert into periodo_letivo(cod_perido, dat_inicial, dat_final)  
    values (5,'01-MAR-96','30-JUN-96');
```

No caso acima onde serão inseridos dados em todas as colunas da tabela, a lista de colunas não precisa ser especificadas, e os dados devem ser colocados na ordem em que aparecem na tabela.

- Valores caracter e de data devem ser especificados entre aspas simples. Valores que não vão ser especificados podem ser definidos como Null.

Inserindo Informações de Data e Hora

- Na inserção de um valor de data, o formato **dd-mon-yy** é normalmente usado. Com este formato o século é usado como 19. O campo data também pode conter a informação de hora, se não for especificado, será assumido o valor 00:00:00.
- Se a data tem que ser entrada com outro século deve ser usada a função **To Date**:

– Ex.:

```
insert into periodo_letivo(cod_perido, dat_inical, dat_final)
values      (6,
             TO_DATE ('01/08/2001','DD/MM/YYYY'),
             TO_DATE ('30/11/2001','DD/MM/YYYY') );
```

Copiando Linhas de outra Tabela

- Você pode usar os dados existentes em uma tabela para inserir dados em outra:

```
insert into table [ ( coluna,coluna,...) ]  
  select lista-de-seleção  
  from tabela;
```

- Ex.: Para copiar todas as informações dos alunos de NATAL para a tabela ALUNOS_NATAL:

```
insert  into alunos_natal  
        ( cod_aluno, nom_aluno, sexo, nom_cidade, ano_ingresso )  
  select cod_aluno, nom_aluno, sexo, nom_cidade, ano_ingresso  
  from   aluno  
  where  nom_cidade = 'NATAL';
```

- Neste caso não é utilizada a cláusula Values.

Alterando Linhas de Tabelas

- O comando update permite ao usuário alterar os valores em linhas da tabela:

```
update table  
set    column [ ,column...] = { expressão, subquery }  
[where condição];
```

– Ex.:

```
update periodo_letivo  
set    dat_inicial = '01-MAR-97',  
       dat_final   = '30-JUN-97'  
where  cod_periodo = 5;
```

Se o where for omitido, todas as linhas na tabela serão alteradas.

Apagando Linhas da Tabela

- O **comando delete** é usado para eliminar linhas de uma tabela:

```
delete from tabela  
[where condição ];
```

- Ex.: Para apagar todas as informações sobre o período letivo 5 da tabela `periodo_letivo`.

```
delete from periodo_letivo  
where cod_periodo = 5;
```

Se o **where** não for especificado, todas as linhas da tabela serão eliminadas.

Apagando Linhas da Tabela

- **Comando truncate:**

- O comando truncate apaga todas as linhas de uma tabela. Diferente do delete, o comando truncate não pode ser desfeito, já que não gera informações de rollback. O comando truncate é um DDL.

Ex.:

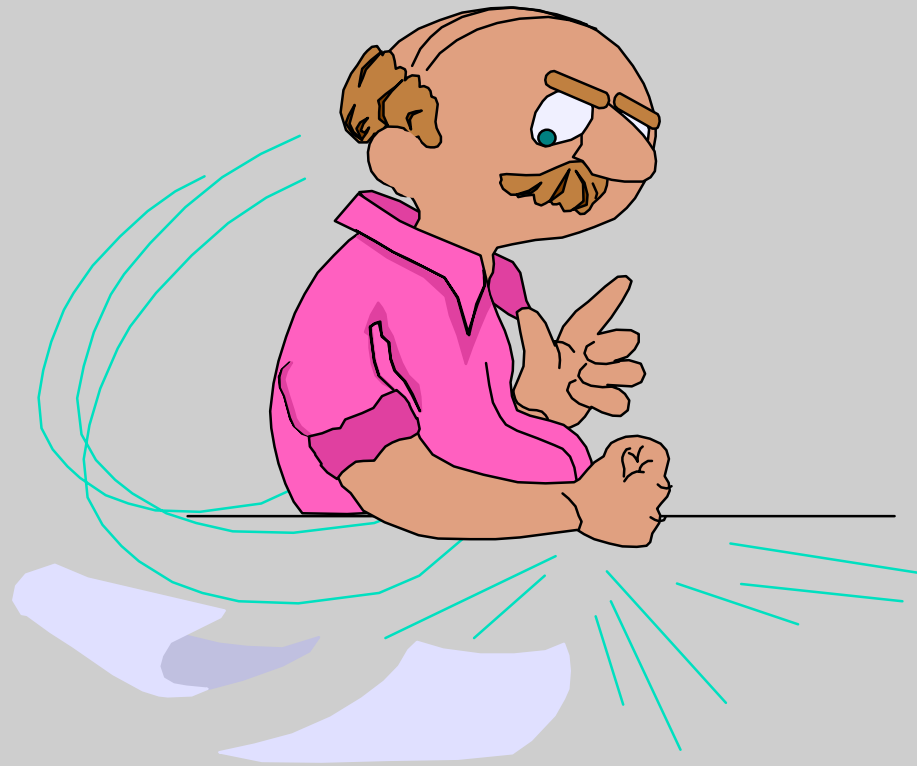
```
SQL> truncate table tabela;
```

Módulo IV

- ⇒ Gerenciando Objetos do Oracle7
- ⇒ O Dicionário de Dados Oracle

Introdução ao

Oracle7



Gerenciando Objetos do Oracle7

Gerenciando Objetos do Oracle7

- **Objetivos desta unidade:**
 - Introduzir os conceitos básicos de gerenciamento de objetos do Oracle7. Os objetos abordados serão:
 - Tabelas
 - Visões
 - Índices
 - Sinônimos
 - Seqüências
 - Introduzir os principais comandos DDL (Data Definition Language) necessários para criar/alterar/remover objetos do Oracle7

Estrutura de Dados Oracle

- Tabelas podem ser criadas a qualquer momento;
- O tamanho dos dados é variável, somente os números e/ou caracteres são realmente guardados no banco;
- A estrutura da tabela pode ser modificada on-line;
- Não é preciso especificar o tamanho de uma tabela.

Criação de Tabelas

- **O nome de uma tabela deve seguir as regras padrões de nomes de objetos Oracle:**
 - O nome deve começar com uma letra;
 - Ele pode conter letras, números e o caracter especial (_ underscore) . Podem também ser usados os caracteres (\$ e #) (o que não é recomendado) ;
 - O nome de uma tabela não é sensível a letras maiúsculas / minúsculas. A tabela emp pode ser referenciada por: aluno, ALUNO, Aluno, etc.;
 - Deve ter até no máximo 30 caracteres;
 - O nome da tabela tem que ser único (incluindo outros objetos do banco de dados);
 - O nome não pode ser uma palavra reservada de SQL

Tipos das Colunas

- Quando uma tabela é criada, você precisa especificar o tipo de dados para cada coluna da tabela.
- Tipos de dados em banco de dados Oracle:

Datatype	Descrição	Tamanho máximo
char (<i>tamanho</i>)	campo caracter tamanho fixo	255
varchar2(<i>tamanho</i>)	campo caracter tamanho variável (tamanho máximo tem que ser especificado)	2000
number (<i>p</i> , <i>e</i>)	Campo numérico de tamanho variável	38,38
date	tamanho fixo no formato data (DD-MON-YY)	-
Long	campo caracter de tamanho variável	2Giga
raw	campo binário de tamanho variável (tamanho máximo deve ser especificado)	2000
long raw	campo binário de tamanho variável (tamanho máximo deve ser especificado)	2Giga
ROWID	campo binário que indica o endereçamento das linhas de uma tabela	06

Criando uma Tabela

- Para criar uma tabela no SQL*Plus, usa-se o comando `create table`:

```
create table nome_da_tabela
( nome_de_coluna tipo( tamanho) [NULL | NOT NULL ],
  nome_de_coluna tipo( tamanho) [NULL | NOT NULL ],
  nome_de_coluna tipo( tamanho) [NULL | NOT NULL ], ...);
```

- **Ex.:**

```
create table periodo_letivo2
(cod_periodo number(3) not null,
 dat_inicial date      not null,
 dat_final   date);
```

- Após o comando ser executado o Oracle retorna a seguinte mensagem:

```
Table created.
```

Criando uma Tabela

- Para ver a descrição da estrutura da tabela, entre com o seguinte comando:

```
SQL> desc periodo_letivo
```

Name	Null?	Type
-----	-----	-----
COD_PERIODO	NOT NULL	NUMBER(3)
DAT_INICIAL	NOT NULL	DATE
DAT_FINAL		DATE

Cláusula Constraint

- Uma restrição de integridade (integrity constraint) é uma regra que restringe os dados de uma ou mais colunas de uma tabela garantindo o relacionamento entre elas. Por exemplo, uma constraint de integridade que existe entre as tabelas ALUNO e MATRICULA força que uma linha só possa ser incluída na tabela MATRICULA se o aluno informado na inclusão existir na tabela ALUNO.

– Ex.:

```
SQL> insert into matricula
  2  (cod_perodo, cod_curso, cod_aluno, dat_matricula, val_nota_final)
  3  values (1, 1, 10, '01-JAN-95', null);
ERROR at line 1:
ORA-02291: integrity constraint (SYSTEM.COD_ALUNO_FK) violated - parent key
not found
```

- As constraints podem ser criadas para tabelas e para colunas das tabelas e são especificadas nos comandos create ou alter table.

Cláusula Constraint

- O objetivo de uma constraint é criar uma série de restrições para os valores que são colocados no banco de dados. Estas restrições são verificadas todas as vezes que são usados os comandos: insert, delete e update.
- Constraints são utilizadas para:
 - forçar a entrada de valores em colunas **Not Null**;
 - especificar que determinada coluna só pode ter valores únicos em relação às linhas da tabelas;
 - identificar colunas como primary key;
 - estabelecer restrições de foreign key;
 - checar o valor entrado na coluna conforme valor pré-determinado: check.

Cláusula Constraint

- **Table constraints:**

- **Table constraints** são restrições que fazem parte da definição global da tabela e se refere a uma ou mais colunas da tabela:

```
create table exemplo
( e1 number(4),
  e2 number(4),
  primary key (e1, e2) );
```

- **Columns constraints:**

- **Columns constraints** são restrições específicas de uma coluna:

```
create table exemplo2 (
e1 number(2) primary key,
e2 varchar2(5) NOT NULL, ...);
```

- Constraints que estabelecem restrições sobre mais de uma coluna só podem ser declaradas como **Table Constraint**; as que restringem apenas uma coluna podem ser tanto **Table/Column Constraint**.

Cláusula Constraint

– Exemplos de constraints:

```
CREATE TABLE MATRICULA
(COD_PERIODO      NUMBER(3)      NOT NULL,
 COD_CURSO        NUMBER(3)      NOT NULL,
 COD_ALUNO        NUMBER(3)      NOT NULL,
 DAT_MATRICULA    DATE           NOT NULL,
 VAL_NOTA_FINAL   NUMBER(5,2),
 CONSTRAINT MATRICULA_PK PRIMARY KEY (COD_PERIODO, COD_CURSO, COD_ALUNO),
 CONSTRAINT VAL_NOTA_FINAL_CK CHECK (VAL_NOTA_FINAL BETWEEN 0 AND 100),
 CONSTRAINT COD_PERIODO_FK
 FOREIGN KEY (COD_PERIODO) REFERENCES PERIODO_LETIVO (COD_PERIODO),
 CONSTRAINT COD_CURSO_FK
 FOREIGN KEY (COD_CURSO) REFERENCES CURSO (COD_CURSO),
 CONSTRAINT COD_ALUNO_FK
 FOREIGN KEY (COD_ALUNO) REFERENCES ALUNO (COD_ALUNO));
```

Cláusula Constraint

– Outra sintaxe para criar a mesma tabela:

```
CREATE TABLE MATRICULA
(COD_PERIODO      NUMBER(3)  NOT NULL REFERENCES PERIODO_LETIVO (COD_PERIODO),
 COD_CURSO        NUMBER(3)  NOT NULL REFERENCES CURSO (COD_CURSO),
 COD_ALUNO        NUMBER(3)  NOT NULL REFERENCES ALUNO (COD_ALUNO),
 DAT_MATRICULA    DATE        NOT NULL,
 VAL_NOTA_FINAL   NUMBER(5,2) CHECK (VAL_NOTA_FINAL BETWEEN 0 AND 100),
 CONSTRAINT MATRICULA_PK PRIMARY KEY (COD_PERIODO, COD_CURSO, COD_ALUNO));
```

Alterando uma Tabela

- Através do comando `alter table` o usuário pode alterar a estrutura de uma tabela, adicionar ou retirar uma constraint, etc.

- Adicionar uma coluna:

```
alter table aluno add (dat_nascimento date);
```

- Modificar um tipo/tamanho de um campo:

```
alter table aluno modify (nom_aluno varchar2(70));
```

- Adicionar uma constraint:

```
alter table aluno add(check(sexo in ('M','F')));
```

Removendo / Renomeando uma Tabela

- **Para remover uma tabela da base de dados utilize o comando Drop Table:**

```
drop table aluno;
```

- **Ao deletar uma tabela:**
 - Todos os dados da tabela serão perdidos junto com os índices associados;
 - Qualquer View, ou sinônimos existentes continuarão existindo porém inválidos;
 - Somente o criador da tabela e o DBA podem eliminá-la.
- **Para renomear uma Tabela, View, ou Sinônimos**

```
rename nome_antigo to nome_novo
```

- Para trocar o nome da tabela emp:

```
rename aluno to alunos;
```

- Em Oracle, não existe comando para remover colunas de tabelas

Removendo / Renomeando uma Tabela

- **Mudanças que não podem ser feitas:**
 - Mudar uma coluna que contém campos nulos para **Not Null**;
 - Adicionar uma coluna **Not Null** em tabela que não esteja vazia (adicione a coluna, preencha e troque para **Not Null**);
 - Diminuir o tamanho ou o tipo de uma coluna a não ser que ela esteja vazia.

Criando uma Tabela através de um Comando Select

- **Uma tabela pode ser criada aproveitando a estrutura e os dados de uma tabela já existente:**
 - Ex.:

```
create table aluno_copia
as
select * from aluno;
```
- **No exemplo dado a tabela aluno_copia terá a estrutura idêntica à da tabela aluno.**

Criando uma Tabela através de um Comando Select

- Podem ser criadas ainda tabelas com alguns campos e/ou dados da tabela original.

– Ex.:

```
create table aluno_copia  
as  
select cod_aluno,nom_aluno from aluno  
where sexo = 'M';
```

Uma maneira de criar uma nova tabela apenas com a estrutura da tabela já existente é colocar na cláusula **where** uma condição que fará com que a Query não retorne nenhuma linha.



Views

Views

- Definição: Uma view (visão) é uma forma pré-determinada de visualizar dados de uma ou mais tabelas como se fosse uma única tabela.
- Uma view é um subconjunto de uma outra tabela ou view que pode ser manipulada como uma tabela.
- Uma view não existe fisicamente como uma tabela. Apenas comando select que define a view que é guardado no banco de dados.
- Views são usadas para diversas situações:
 - restringir o acesso aos dados da tabela original, já que a view pode conter apenas algumas colunas de uma tabela.
 - simplificar para usuários a estrutura de tabelas, ou comandos selects mais complicados.
 - permitir usuários diferentes visualizar os dados de forma diferente.

Criação de Views

- **Sintaxe:**

```
CREATE VIEW nome_da_view
           [(coluna1,coluna2,....)]
AS
SELECT      parâmetros do comando select.....
```

- **Ex.:**

- Para criar uma view referente à tabela aluno mas apenas com os alunos da cidade de NATAL:

```
SQL> create view v_alunos_natal
2  as select *
3  from aluno
4  where nom_cidade = 'NATAL';
```

View created.

Criação de Views

- Para selecionar todos os dados da view criada:

```
SQL> select * from v_alunos_natal;
```

COD_ALUNO	NOM_ALUNO	S	NOM_CIDADE	ANO_INGRESSO
4	FELIPE	M	NATAL	1993
5	ANA	F	NATAL	1995

- **OBS**

- A cláusula ORDER BY não pode ser utilizada em views.

Criação de Views

- Views que se referem a uma única tabela e não contém funções de dados (data, character ou numéricas) e nem grupos são chamadas de views simples.
- Views complexas são views que se referem a mais de uma tabela e podem conter funções de grupo ou funções de dados.
 - Para criar uma view contendo funções de grupo e dados de duas tabelas:

```
SQL> create    view media_aluno
  2              (Nome, Media)
  3 as
  4 select      nom_aluno, avg(val_nota_final)
  5 from        matricula m, aluno a
  6 where       m.cod_aluno = a.cod_aluno
  7 group      by nom_aluno;
```

View created.

Criação de Views

Foram especificados nomes para as colunas da view já que no comando select existem funções e o nome para a coluna da view seria inválido, ex.: avg(val_nota_final) não é um nome de coluna válido. Poderia ser usado também o alias para as colunas no comando select . Se os dois forem especificados vale o especificado para a view.

Alterando Dados através de uma View

- A deleção de dados através de uma view não é permitida quando:
 - a view foi criada através de um join;
 - contém funções de grupo;
 - contém a cláusula distinct;
- O update não é permitido se:
 - a view foi criada através de um join;
 - contém funções de grupo;
 - contém a cláusula distinct;
 - contém colunas definidas por expressões, ex.: carga_horaria/4;
- O insert não é permitido quando:
 - a view foi criada através de um join;
 - contém funções de grupo;
 - contém a cláusula distinct;
 - contém colunas definidas por expressões, ex.: carga_horaria/4;
 - alguma coluna Not Null não foi selecionada pela view.

Eliminando uma View

- Para eliminar uma view usa-se o comando drop view.

- Ex.:

```
SQL> DROP VIEW nome_view
```

```
View dropped;
```

Nenhum dado referenciado pela view é afetado, já que eles estão guardados fisicamente nas tabelas da base e não na view.

- Para verificar as views existentes e o SQL que compõe a views do usuário ele pode selecionar a tabela: User_Views.



Sinônimos e Seqüências

Criação de Sinônimos

- Para acessar ou alterar uma tabela de um outro usuário, ela deve ser referenciada com o prefixo do nome do usuário que a criou seguido de um ponto.
- Para o usuário aluno1 fazer um select na tabela CURSO do usuário aluno2 ele deve usar o seguinte comando:

```
select * from aluno2.curso;
```

- O usuário que está fazendo a query na tabela de outro usuário pode criar um sinônimo para acessá-la de forma mais simples:
 - Ex.:
 - Para o usuário aluno1 referenciar a tabela do aluno2 somente como CURSO2, o aluno1 deve executar o seguinte comando:

```
SQL> create synonym curso2 for aluno2.curso;
```

```
Synonym created.
```

- A partir daí o usuário aluno1 pode acessar a tabela com o nome criado:

```
SQL> select * from curso2;
```

Criação de Sinônimos

- O usuário aluno1 não pode conter nenhum objeto com o mesmo nome que está sendo criado.
- O usuário que está criando um sinônimo sobre tabela/view de outro usuário deve ter acesso a esta tabela.
- O usuário com privilégio de DBA pode criar sinônimos públicos, ou seja, sinônimos que todos os usuários da base de dados têm acesso.

Ex.:

```
SQL> create public synonym curso on escola.curso;  
Synonym created.
```

Sinônimos fornecem um meio conveniente de se obter transparência de localização e independência de dados.

Removendo Sinônimos

- Os sinônimos são removidos pelo comando DROP.

```
drop synonym [nome_do_sinonimo]
```

- Sinônimos não podem ser alterados

Criação de Seqüências Automáticas

- **Seqüências** (sequences), são estruturas do banco de dados que geram números seqüenciais que podem ser usados para gerar chaves únicas ou para gerar seqüências numéricas controladas pelo próprio banco.
 - Para gerar uma sequence no banco de dados:

```
SQL> create sequence SEQ increment by 1
      2 start with 1
      3 maxvalue 10
Sequence created.
```
 - No comando create sequence os operandos são os seguintes:
 - increment by n
 - o valor que será incrementado cada vez que a sequence for selecionada.
 - start with n
 - valor inicial da sequence.
 - maxvalue n
 - valor final da sequence.

Criação de Seqüências Automáticas

- Para a utilização dos números gerados pela sequence é feito um select na sequence, especificando que o usuário deseja o próximo valor gerado:

Ex.:

```
SQL> select seq.nextval from dual;
```

```
NEXTVAL
```

```
-----
```

```
1
```

- Na próxima execução do comando select o valor será incrementado de acordo com a definição da sequence.
- O usuário pode selecionar o varlor corrente da sequence através do select:

```
SQL> select seq.currval from dual;
```

```
CURRVAL
```

```
-----
```

```
1
```

Alterando/Removendo Sequences

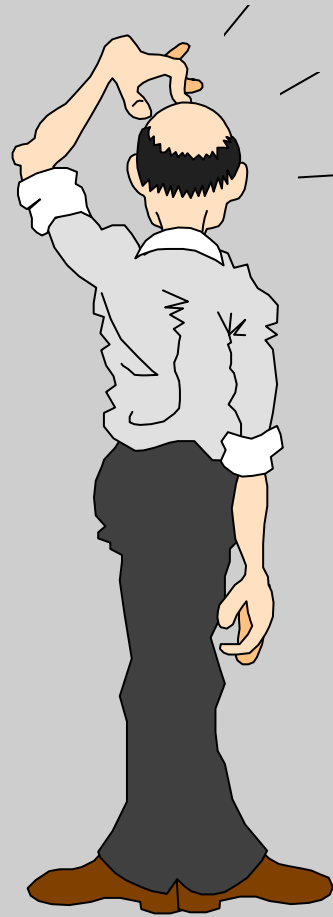
- Para alterar uma sequence utilizamos o comando ALTER SEQUENCE

```
SQL> alter sequence SEQ increment by 1  
2 start with 1  
3 maxvalue 1000
```

- Para eliminar uma sequence utilize o comando drop:

```
SQL> drop sequence seq;  
Sequence dropped.
```

Para alterar o valor de **START WITH**, devemos remover e recriar a sequence.



Criação de Índices

Criação de Índices

- A criação de índices em uma base de dados serve para aumentar a velocidade de queries em tabelas grandes e garantir que tabelas terão valores únicos nas colunas apropriadas.
- Depois de criado, o índice vai ser usado pelo Oracle toda vez que for necessário para aumentar a performance. O próprio Oracle gerencia o índice, dispensando do usuário qualquer manutenção nestes índices.

Tipo	Descrição
unique	garante que o valor na coluna vai ser único.
non unique	garante o acesso mais rápido possível aos dados (default).
single column	o índice é composto de apenas uma coluna.
concatenado	índice composto de até 16 colunas da tabela.

Tipos de Índices

- Criação de um índice:

- Sintaxe:

```
create [unique] index nome_do_índice  
on tabela (coluna1 [ ,coluna2...]) ;
```

- Para criar um índice para melhorar a performance.

```
SQL> create index aluno_indice on aluno (nom_aluno);  
Index created.
```

- Para criar um índice para garantir a unicidade de uma coluna:

```
SQL> create unique index aluno_indice  
2 on aluno (nom_aluno);  
Index created.
```

- Para criar um índice concatenado:

```
SQL> create unique index aluno_indice on aluno (cod_aluno,nom_aluno);  
Index created.
```

- Para remover um índice usa-se o comando drop:

```
SQL> drop index aluno_indice;  
Index dropped.
```



O Dicionário de Dados do Oracle7

O Dicionário de Dados do Oracle7

- O **Dicionário de Dados** é uma das mais importantes partes do **RDBMS** Oracle. Ele consiste em um conjunto de tabelas e views que são um guia de referência sobre a base de dados. O dicionário de dados contém informações como:
 - o nome dos usuários Oracle;
 - os direitos e privilégios que eles têm ;
 - nome dos objetos do banco de dados ;
 - as constraints das tabelas ;
 - etc.
- O **Dicionário de Dados** é criado junto com a criação do banco de dados. As alterações feitas na base de dados são feitas pelo **RDBMS** Oracle.
- O Dicionário de dados é uma referência para todos os usuários do banco de dados. É uma fonte valiosa de informações para os usuários finais, desenvolvedores de aplicações, e DBA's.

Acesso ao Dicionário de Dados

- O acesso ao **Dicionário de Dados** Oracle é feito pelos usuários através de selects nas tabelas e views do dicionário. A alteração destas tabelas e views é feita através do RDBMS, que verifica os acessos e permite ou não as operações no banco. Nenhuma tabela ou view do sistema deve ser alterada manualmente sob o risco de se perder a integridade dos dados que estão contidos na base.

Tabelas e Views do Dicionário de Dados

- As **tabelas** do dicionário de dados são criadas junto com a criação do **banco de dados** e raramente são acessadas diretamente pelo usuário. Para o acesso aos usuários são criadas, views com dados mais resumidos e de forma mais simples (algumas tabelas chegam a ter dezenas de colunas e nomes pouco significativos).
- Os nomes das **views** que são criadas refletem o tipo de uso a que elas se destinam.
- As principais **views** são:

user_XXXXX	mostra informações sobre os objetos <i>do usuário</i> .
all_XXXXX	mostra objetos que o usuário <i>tem acesso</i> além dos objetos do <i>próprio usuário</i> .
dba_XXXXXX	para usuários que tem o acesso de dba, mostra objetos pertencentes à <i>base de dados</i> e objetos que o usuário não dba não tem acesso.
dictionary	contém todos os objetos <i>do dicionário</i> de dados acessíveis pelo usuário.

Tabelas e Views do Dicionário de Dados

- A view Dictionary mostra todos os objetos do dicionário de dados que são acessíveis pelo usuário com um pequeno comentário sobre o conteúdo de alguns destes objetos.
 - Para verificar o conteúdo desta view entre com o comando:


```
SQL> select * from dictionary;
```
- As views do dicionário de dados mais utilizadas são as seguintes:

Nome da view	Sinônimo	Descrição
dictionary	dict	todos os objetos do dicionário de dados acessíveis pelo usuário com uma breve descrição.
user_objects	obj	objetos pertencentes ao usuário
user_tables	tabs	descrição das tabelas do usuário
user_sequences	seq	descrição das sequences do usuário
user_synonym	syn	sinônimos do usuário
user_views		texto das views do usuário
user_indexes	ind	descrição dos índices do usuário
all_objects		objetos acessíveis pelo usuário

Consultando o Dicionário de Dados

- Exemplo 1: Tabelas do usuário corrente

```
select table_name
from   user_tables;
```

- Exemplo 2: Colunas do tipo Date na tabela PERIODO_LETIVO

```
select column_name
from   user_tab_columns
where  table_name = 'PERIODO_LETIVO'
and    data_type = 'DATE';
```

- Exemplo 3: Informações dos objetos do usuário corrente

```
select object_name, object_type, created
from   user_objects;
```

Consultando o Dicionário de Dados

- Exemplo 4: Informações dos objetos visíveis pelo usuário corrente

```
select object_name, object_type, created
from all_objects;
```

- Exemplo 5: Todos os sinônimos visíveis pelo usuário

```
select synonym_name, table_name
from all_synonyms;
```

- Exemplo 6: A definição de uma certa view

```
select text
from user_views
where view_name = 'ALUNOS_NATAL';
```

Consultando o Dicionário de Dados

- Exemplo 7: Consultar o nome da primary key da tabela MATRICULA

```
select constraint_name
from   user_constraints
where  table_name = 'MATRICULA'
and    constraint_type = 'P';
```

- Exemplo 8: Quais colunas compõem a primary key da tabela MATRICULA?

```
select column_name, position
from   user_cons_columns
where  constraint_name = 'MATRICULA_PK';
```

Módulo V

- ⇒ Transações
- ⇒ Leitura Consistente
- ⇒ Mecanismo Lock - Visão Geral
- ⇒ Visão Geral de Usuário e Segurança

Introdução ao

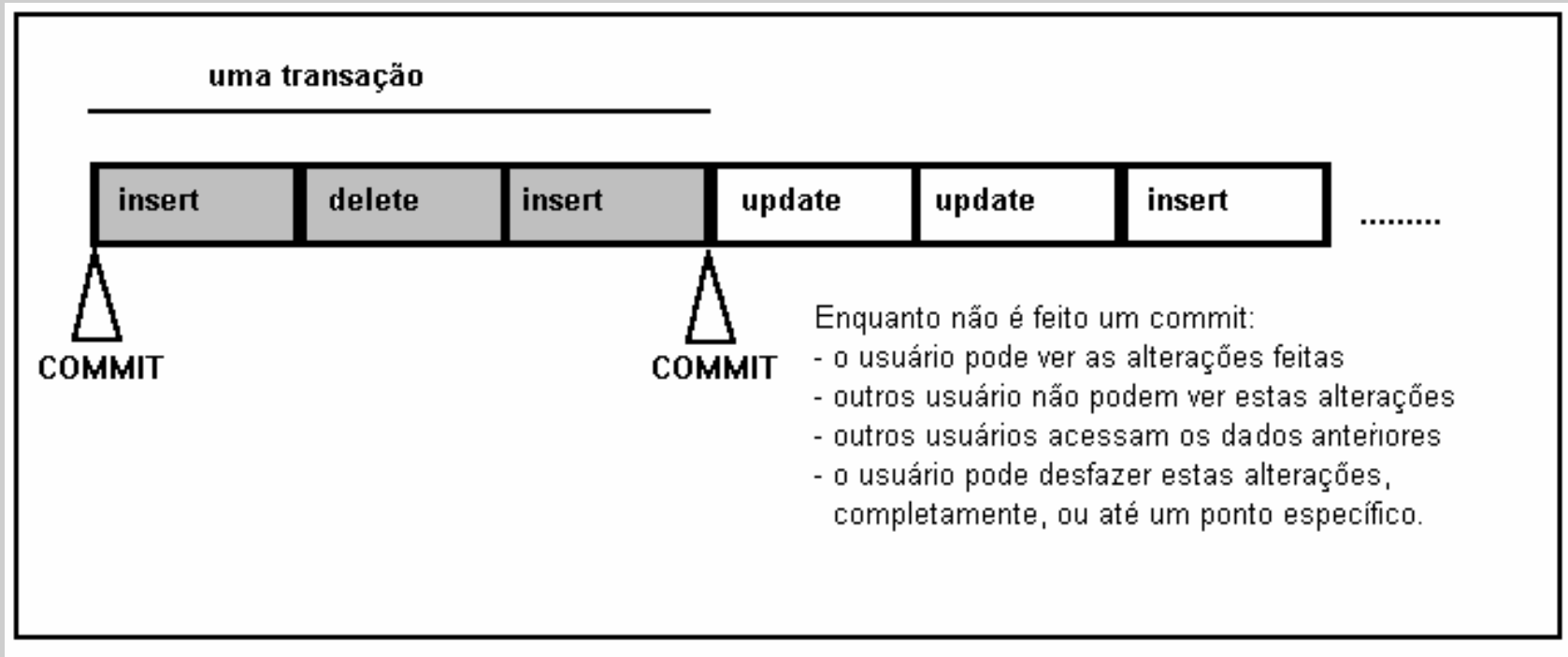
Oracle7



Transações

Transações

- Uma **transação** é uma operação feita no banco de dados que contém uma série de alterações em uma ou mais tabelas.



Transações

- Existem duas classes de transações:
 - **Transações DML** - consiste em uma série de comandos DML e que o Oracle trata como uma unidade lógica de trabalho.
 - **Transações DDL** - contém apenas comandos DDL.
 - Uma transação para o Oracle é tornada permanente para o banco somente no caso de todos os comandos dentro daquela transação terminarem corretamente, se não as alterações são descartadas.
- Uma transação começa quando um comando DML ou DDL é encontrado e termina quando:
 - é executado um comando **Commit** ou **Rollback**;
 - qualquer comando DDL é encontrado;
 - saída do SQL*Plus;
 - falha de máquina.

Transações

- Quando uma transação termina, o próximo comando vai iniciar automaticamente a próxima transação.
- Para tornar alterações permanente, é utilizado o comando Commit. Para desistir das alterações que ainda não foram tornadas permanentes utilize o comando Rollback. O comando Rollback volta os dados até a opção do último Commit feito pelo usuário (explícito, ou implícito).
- Falhas de sistemas:
 - Quando o sistema falha por algum motivo, as alterações que estavam sendo feitas na base e ainda não tinha sido tornadas permanentes também são voltadas ao estado original para garantir a integridade do banco de dados.

Controlando Transações com Comandos SQL

- **Os comandos a seguir são usados para controlar as transações:**
 - Commit [Work]
 - torna as mudanças da transação permanentes;
 - apaga todos os savepoints na transação;
 - termina uma transação;
 - retira os lock's da transação;
 - é feito automaticamente quando:
 - um comando DDL é executado (antes e depois do comando);
 - na desconexão normal do banco de dados;
 - Savepoint nome_do_savepoint
 - é utilizado para dividir uma transação em partes menores;
 - se utilizado um segundo savepoint com o mesmo nome o primeiro é descartado;
 - o número máximo default de savepoints por processo é de 5. (pode ser alterado);

Controlando Transações com Comandos SQL

- Rollback [Work] to [Savepoint] nome_do_savepoint
 - usado para desfazer alterações no banco de dados;
 - pode ser usado para desfazer alterações até um determinado ponto (savepoint);
 - se usado sem o savepoint:
 - termina a transação;
 - desfaz todas as alterações na transação corrente;
 - desfaz os locks da transação;
- Exemplo do controle de transações que pode ser feito:

```
SQL> insert into periodo_letivo values (10,'01-JAN-97', '01-JAN-98');
1 row created.
SQL> savepoint insert_feito;
Savepoint created.
SQL> update dept set dat_final = '01-JAN-99';
SQL> rollback to insert_feito;
Rollback complete.
SQL> commit;
Commit complete.
```

Controlando Transações com Comandos SQL

– Commit automático:

- Os comandos commit e rollback podem ser usados manualmente (como já foi visto) ou automaticamente através do comando SQL*Plus Autocommit:

– Pode ser usado de duas maneiras:

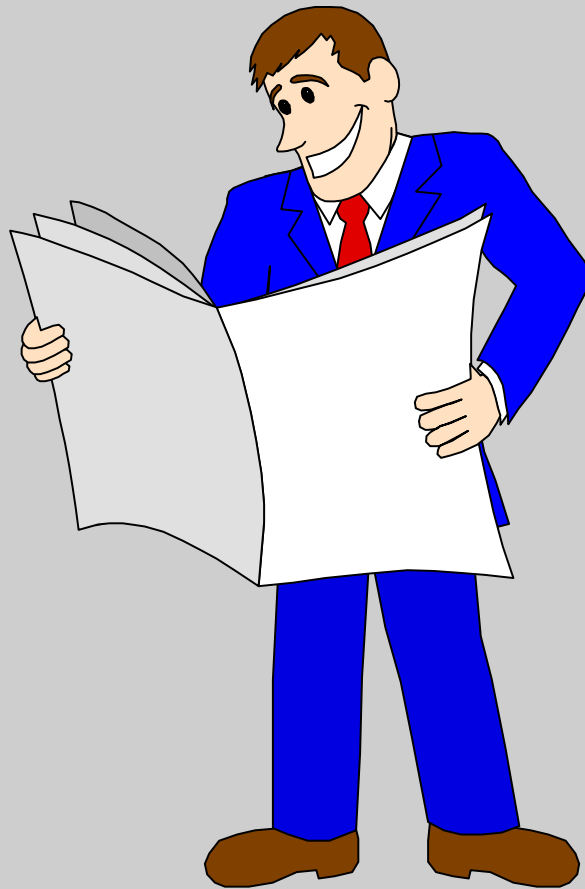
SET AUTO[COMMIT] ON	Um commit é feito a cada comando INSERT, UPDATE ou DELETE.
SET AUTO[COMMIT] OFF	Opção default - um commit é usado explicitamente ou quando são utilizados comandos DDL e na saída normal do SQL*Plus.

– Para verificar se o autocommit está ligado ou não use:

```
SQL> show autocommit;
```

```
autocommit OFF
```

```
SQL>
```



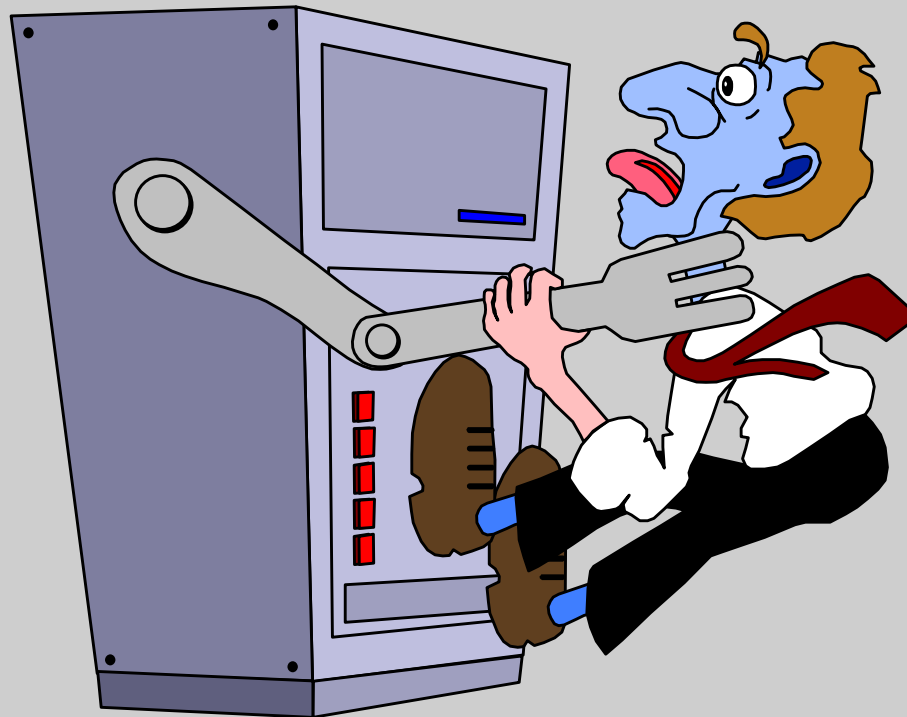
Leitura Consistente

Leitura Consistente

- **Os usuários do Banco de dados fazem dois tipos de acesso ao Banco de dados:**
 - Acesso somente para leitura (select);
 - Acesso para escrita (insert update e delete).
 - O banco de dados deve garantir para os leitores do banco a consistência dos dados que ele está vendo. Os leitores não podem ver dados que estão sendo alterados naquele momento, pois comprometeria a consistência destes dados.
 - Da mesma forma os escritores do banco de dados precisam da garantia que os dados que ele está modificando estão consistentes e não estão sendo modificados por outro escritor.
 - A leitura consistente é implementada mantendo uma cópia parcial dos dados nos segmentos de rollback

Leitura Consistente

- Quando uma alteração é feita no banco de dados, o banco mantém a cópia dos dados antes da alteração no segmento de rollback. Quando um outro usuário for acessar estes dados ele terá a visão dos dados que estão no segmento de rollback.
- Quando o escritor faz o commit da sua transação o banco de dados altera definitivamente estes dados no banco de dados e elimina os dados anteriores do segmento de rollback. A partir daí os outros usuários do banco passam a ter acesso a estes dados já com as alterações.
- Se o escritor faz um rollback, os dados anteriores são copiados novamente para o banco do segmento de rollback , retornando a versão anterior dos dados.



Mecanismo de Lock - Visão Geral

Mecanismo de lock - Visão Geral

- Uma das principais tarefas de um **RDBMS** é controlar o acesso simultâneo de diversos usuários (aplicações) ao banco de dados. Esse controle é necessário para garantir que nenhum usuário estará alterando um registro que está sendo alterado ou foi apagado por outro usuário, mantendo assim a integridade dos dados do Banco.
- O **Oracle** permite este tipo de concorrência através dos mecanismos de Lock.
- Fazer um **Lock** de uma tabela é parte essencial do mecanismo de consistência e integridade de sua base de dados.

O que é um Lock?

- O **Lock** é um mecanismo usado para controlar o acesso dos dados em um sistema multi-usuário. Ele previne que o mesmo dado seja alterado por dois usuários, e que uma tabela que esteja sendo alterada em sua estrutura tenha seus dados alterados.
- Um **Lock** é feito quando um dado está sendo alterado, ou quando a estrutura de uma tabela está sendo alterada. Ele é finalizado quando um commit ou rollback é feito no dado que estava sendo alterado, ou quando a transação é terminada.

Níveis de Locks

- **Existem dois tipos de Locks:**

- Table Lock (Locks em Tabelas) - quando toda a tabela está em lock para um usuário;

```
SQL> lock table aluno in exclusive mode;  
Table(s) Locked.
```

- Row Lock (Lock de Linha) - quando uma linha está em lock por causa da alteração dos dados desta linha.

```
SQL> update aluno  
2 set ano_ingresso = 1990  
3 where nom_aluno='MARIA';  
1 row updated.
```

```
SQL>
```

- Esta linha continuará em lock até o final da transação.

Tipos de Locks

- Exclusive (x)
 - Recurso bloqueado somente para um usuário. Somente este usuário pode alterar os dados . Os outros usuários só podem ler os dados.
- Row Share(RS)Share update
 - Outros usuários têm acesso aos dados, mas nenhum usuário pode fazer um lock table na tabela.

Deadlocks

- **Deadlocks** ocorrem quando dois usuários estão tentando acessar para alteração os mesmos dados ao mesmo tempo. Tipicamente, eles estão esperando cada um por um recurso que está sendo bloqueado pelo outro.

– EX.:

Tempo	Primeira Transação	Segunda Transação
1	<pre>update aluno set nom_cidade='NATAL' where nom_aluno = 'ANA';</pre>	<pre>update periodo_letivo set dat_final='01-MAR-92' where cod_periodo = 1;</pre>
2	<pre>update periodo_letivo set dat_final='01-MAR-90' where cod_periodo = 1;</pre>	<pre>update aluno set ano_ingresso= 1995 where nom_aluno = 'ANA';</pre>

- Quando o Oracle detecta um deadlock, ele envia uma mensagem de erro para um dos usuários e faz um rollback de seu comando. Normalmente o usuário deve então fazer um rollback de sua transação. Se possível o usuário pode aguardar alguns instantes e tentar novamente a operação.



Visão Geral de Usuários e Segurança

Visão Geral de Usuários e Segurança

- **Objetivos desta unidade:**

- Esta unidade mostra como criar usuários e atribuir privilégios de acesso e alterações na base de dados.

- **Criação de usuários Oracle:**

- Para acesso ao banco de dados, é preciso informar um usuário e password. Este usuário é criado pelo usuário administrador do banco de dados (system) ou outro usuário que tenha o acesso de DBA.

- Para criar um usuário Oracle:

```
SQL> create user aluno1 identified by aluno1  
2     default tablespace curso  
3     temporary tablespace temp;
```

```
User created.
```

Atribuindo Permissões e Direitos aos Usuários

- Assim que um usuário é criado no banco ele não tem privilégios de acesso a conexão no banco nem de fazer nenhuma tarefa no banco de dados como criar tabelas, views, etc.
- Para isso o administrador do banco (DBA), deve atribuir ao usuário estes acessos (Roles).

Atribuindo Permissões e Direitos aos Usuários

- Existem 05 roles default no banco de dados:

Role	Direitos
connect	se conectar ao banco de dados.
resource	criar tabelas, sinônimos, views, etc. Acesso só aos próprios objetos.
imp_full_database	usar o comando imp para recuperar a base de dados de um backup.
exp_full_database	fazer backup da base de dados completa.
dba	administração do banco e acesso aos objetos de outros usuários.

Troca de Password de um Usuário

- A troca de password de um usuário só pode ser feita pelo próprio usuário ou pelo DBA.
 - Ex.:
 - Para trocar a password do usuário aluno1 para aluno:

```
SQL> alter user aluno1 identified by aluno;  
User altered.
```

Privilégios em Tabelas

- Quando um usuário cria um objeto no banco de dados, só ele tem acesso àquele objeto. O dono deste objeto pode passar determinado privilégio para que outros usuários possam também acessá-lo.
- Para isso o usuário deve usar o comando grant.

– Sintaxe:

```
Grant    privilégio  
on       objeto  
to       [ usuário | role ] ;
```

Privilégios em Tabelas

- Privilégios sobre objetos que podem ser passados para outros usuários/roles são:

Privilégio:	Direitos sobre o objeto:
select	selecionar dados da tabela ou view.
insert	inserir linhas na tabela ou view.
update	alterar linhas na tabela ou view.
delete	eliminar linhas da tabela ou view.
alter	alterar a estrutura de de uma tabela.
index	indexar uma tabela.
references	fazer referência à tabela em constraints de foreing key.
all	todos os privilégios sobre o objeto

Privilégios em Tabelas

– Ex.:

– Para dar o direito de fazer select e insert na tabela curso do usuário aluno1 para o aluno2;

```
SQL> grant select,insert on curso to aluno2;  
Grant succeeded.
```

– Para dar todos os direitos sobre a tabela para o usuário aluno2:

```
SQL> grant all on curso to aluno2;
```

- Os privilégios passados para um usuário, como no exemplo acima, não podem ser passados para outros usuários por quem recebeu o direito. Para isso usa-se a cláusula with grant option:

• Ex.:

```
SQL> grant all on curso to aluno2 with grant option;
```

- Desta forma o usuário aluno2 agora pode passar os privilégios recebidos sobre a tabela CURSO para outros usuários.
- Um usuário pode também tornar um objeto público, ou seja de acesso a todos os usuários da base de dados:

• Ex.:

```
SQL> grant all on curso to public;
```

Retirando os Privilégios de um Objeto

- Para retirar um privilégio dado sobre um objeto é usado o comando revoke:

Sintaxe:

```
SQL> revoke privilégio
      on      objeto
      from    [ usuário| role ];
```

- Quando você retira um privilégio de um objeto que foi dado a um usuário, este usuário perde o privilégio e todos os usuários a quem ele deu o privilégio também.

– Ex.:

```
SQL> revoke all
2     on      curso
3     from    aluno2;
```

- Para saber quais usuários têm privilégios em suas tabelas, views ou sequences, verifique a view `User_Tab_Grants`.