Sumário

[SQL 5](#_Toc79153564)

[Instalar 5](#_Toc79153565)

[Configurar 5](#_Toc79153566)

[Seção 1- Criação das Tabelas e Comandos Iniciais 6](#_Toc79153567)

[Seção 2 -> Tipos de Dados 7](#_Toc79153568)

[Seção 4 – Funções Básicas 7](#_Toc79153569)

[Single-Row 7](#_Toc79153570)

[Datas 7](#_Toc79153571)

[#Diferença de dias é considerado também as horas. Fazer o trunc no resultado. 7](#_Toc79153572)

[--Somando horas em uma data 7](#_Toc79153573)

[--Somar minutos 8](#_Toc79153574)

[--Somar Segundos 8](#_Toc79153575)

[--Proxima data a partir de uma dia da semana 8](#_Toc79153576)

[--Ultimo dia do mes 8](#_Toc79153577)

[--Primeiro dia do mes 8](#_Toc79153578)

[--Primeiro dia do proximo mês 8](#_Toc79153579)

[---Formatação de data 8](#_Toc79153580)

[--Busca o nome do mês 8](#_Toc79153581)

[--Busca o nomed do dia do mês 8](#_Toc79153582)

[--Nome, em extenso, do Ano 8](#_Toc79153583)

[--Concatenar String no To\_char . 8](#_Toc79153584)

[--Convertendo números no TO\_CHAR 8](#_Toc79153585)

[Seção 7 – SubConsultas 9](#_Toc79153586)

[Exemplos 1: 9](#_Toc79153587)

[Exemplo 2: 9](#_Toc79153588)

[Exemplo 3: 9](#_Toc79153589)

[Seção 8: Variáveis de Substituição e SavePoint 9](#_Toc79153590)

[Seção 10 – Constraints: 10](#_Toc79153591)

[--Desabilitando/Habilitando constraint 10](#_Toc79153592)

[--Excluir Constraint 10](#_Toc79153593)

[Seção 11 – Views (Visões) 10](#_Toc79153594)

[Ex: 10](#_Toc79153595)

[--View com parametros de saida 10](#_Toc79153596)

[--View Complexa 10](#_Toc79153597)

[--OPERACAO DML NA VIEW 11](#_Toc79153598)

[--View somente leitura (Nao permite DML) 11](#_Toc79153599)

[Seção 13 - Sequence 11](#_Toc79153600)

[--Valor Atual 11](#_Toc79153601)

[PL/SQL 12](#_Toc79153602)

[1 – Corpo de um bloco Anônimo, Procedure e Função. 12](#_Toc79153603)

[Exemplo Procedure: 12](#_Toc79153604)

[Declarando variáveis: 12](#_Toc79153605)

[Tipos de Variáveis: 12](#_Toc79153606)

[Long 12](#_Toc79153607)

[Long Row 12](#_Toc79153608)

[Binary\_float 12](#_Toc79153609)

[Binary\_double 13](#_Toc79153610)

[Binary\_integer 13](#_Toc79153611)

[PLS\_INTEGER 13](#_Toc79153612)

[Atributo %Type 13](#_Toc79153613)

[Boolean 13](#_Toc79153614)

[Record 13](#_Toc79153615)

[Collection 13](#_Toc79153616)

[Lob 13](#_Toc79153617)

[Variáveis não PL/SQL 13](#_Toc79153618)

[2 – Exemplo utilizando o Dbms\_Output.Put\_Line 13](#_Toc79153619)

[Bind variable 13](#_Toc79153620)

[3 – Quantidade de Linhas Afetadas: 14](#_Toc79153621)

[4 – Loops 14](#_Toc79153622)

[Loop 14](#_Toc79153623)

[For Loop 14](#_Toc79153624)

[While Loop 15](#_Toc79153625)

[5 – Records 15](#_Toc79153626)

[Ex 1: 15](#_Toc79153627)

[Ex 2: 15](#_Toc79153628)

[6 – Collection 16](#_Toc79153629)

[Associative Arrays (Index-by-tables)-> Tamanho variável 17](#_Toc79153630)

[Ex 1: 17](#_Toc79153631)

[Nested Tables -> Tamanho variável 17](#_Toc79153632)

[Ex 1: 17](#_Toc79153633)

[Ex 2: 18](#_Toc79153634)

[Varrays -> Semelhante a matriz. 18](#_Toc79153635)

[7 – Cursores Explícitos 19](#_Toc79153636)

[Exemp 1 – Cursor para recuperar campos: 19](#_Toc79153637)

[Exemp 2 – Cursor para recuperar vários campos: 19](#_Toc79153638)

[Exemp 3- Cursos passando parâmetro 20](#_Toc79153639)

[Exemp 4- Cursos travando tabela 20](#_Toc79153640)

[8 – Tratamento de Erros (Exceções): 20](#_Toc79153641)

[Exemplo 1 – Exceções por padrão: 20](#_Toc79153642)

[Exemplo 2 – Exceções desenvolvidas manualmente 21](#_Toc79153643)

[9 – Procedures 21](#_Toc79153644)

[Exempl 1: 21](#_Toc79153645)

[Exemplo 2: 22](#_Toc79153646)

[Exemplo 3 23](#_Toc79153647)

[10) Functions 23](#_Toc79153648)

[Exemplo 1 – função básica 23](#_Toc79153649)

[Exemplo 2 – Pipelined Table Function 24](#_Toc79153650)

[11 – Packages 24](#_Toc79153651)

[Exemplo 1: 25](#_Toc79153652)

[12 - Triggers 27](#_Toc79153653)

[Exemplo 1 – Trigger Simples. Nível de Comando 27](#_Toc79153654)

[Exemplo 2 – Trigger com condição específica de um campo. Nível de Comando 27](#_Toc79153655)

[Exemplo 3 – Trigger Nível de linha para determinado campo. 28](#_Toc79153656)

[Exemplo 4 – Trigger nível de Linha para toda a Tabela. 28](#_Toc79153657)

[Exemplo 5 – Trigger INSTEAD OF 28](#_Toc79153658)

[13 – Avançado Group by – RollUp, Cube. Rank, Dense\_Rank\_Ration\_to\_Report, LED, LAG 28](#_Toc79153659)

[Ex 1: Usando RollUp no Group By – Serve para criar as subcategorias 29](#_Toc79153660)

[Ex 2: Group By Grouping Sets -> Retorna apenas os SubTotais 29](#_Toc79153661)

[Ex 3: Rank () -> Usar o Rankeamento de posição. Primeiro, Segundo , Terceiro 29](#_Toc79153662)

[Ex 4: Rank com PARTITION BY -> Faz o Ranqueamento considerando determinado Campo 30](#_Toc79153663)

[Ex 5: Dense Rankin -> Não deixa empatado 30](#_Toc79153664)

[Ex 6: RATIO\_TO\_REPORT -> Serve para mostrar o Percentual 31](#_Toc79153665)

[Ex 7: LAG e LEAD – 31](#_Toc79153666)

[14 – Insert All e Merge 31](#_Toc79153667)

[15- Sql Dinâmico 32](#_Toc79153668)

[PL/SQL AVANÇADO 32](#_Toc79153669)

[1 Permissionamento para Debugar. 32](#_Toc79153670)

[2 Record e Collections 33](#_Toc79153671)

[Record -> É como se fosse um Array. 33](#_Toc79153672)

[Collections -> Gerenciamento de múltiplas linhas de dados 33](#_Toc79153673)

[Associative Arrays 33](#_Toc79153674)

[Nested Tables 33](#_Toc79153675)

[VARRAYS 33](#_Toc79153676)

[3 – Query LEVEL para gerar uma lista 33](#_Toc79153677)

[4 – Full Text Search 34](#_Toc79153678)

[5 – Estatísticas 35](#_Toc79153679)

[6 – Views Materializadas 36](#_Toc79153680)

[Utilizamos elas para fazermos cálculos, armazenamentos de dados e dar agilidade na troca de informações entre um banco de dados ou entre tabelas. Este recurso é muito utilizado em ambientes de Data Warehouse, que trabalha com uma enorme quantidade de informações. 36](#_Toc79153681)

[A View Materializada deverá utilizar os dados imediatamente na query rewrite (Seu SELECT), desde modo os dados serão processados com mais agilidade. 36](#_Toc79153682)

[É necessário ter uma VIEW LOG nela. 36](#_Toc79153683)

[7 – Indíces 36](#_Toc79153684)

[8 – FlahsBack 37](#_Toc79153685)

[9 – Table Temporária 37](#_Toc79153686)

[10 – Buscar todos os dicionários do Oracle 37](#_Toc79153687)

[11 – Valores DEFAULT 37](#_Toc79153688)

[12 – Multiples Inserts 37](#_Toc79153689)

[Inser ALL 37](#_Toc79153690)

[Inser First -> 38](#_Toc79153691)

[Insert ALL com Pivot 38](#_Toc79153692)

[13 – With 38](#_Toc79153693)

# SQL

## Instalar

Oracle 11xe.

Ir em Setup.exe -> Executar como Administrador -> Escolher a senha ->

SqlTools

## Configurar

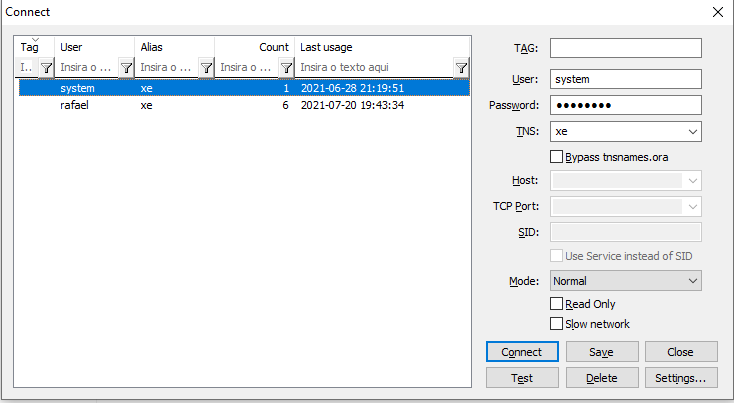
Oracle Cliente

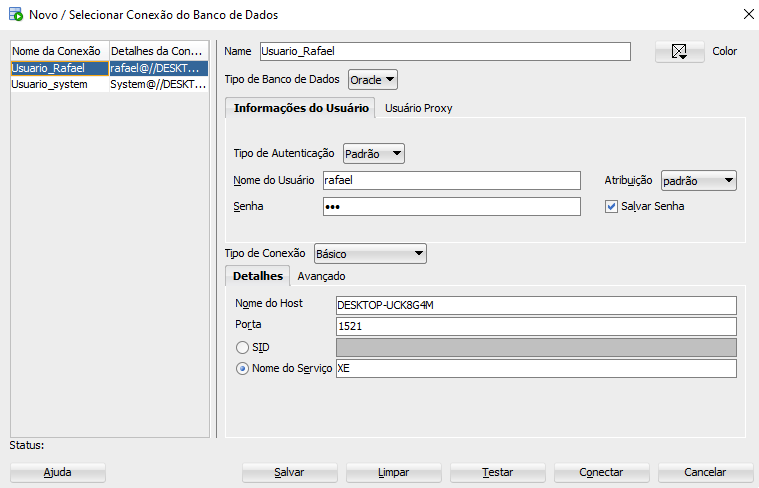
Copiar o Client e ir em C://OracleExe/Client

Networl/admin/tnsadmin.ora -> tem que criar esse arquivo para o SqlTools conseguir acessar o banco.

Criar uma variável de ambiente, em Path, contendo o caminho do cliente(C://OracleExe/Client).

Após isso, em SqlTools , só inserir as informações para conectar ao banco.





Ambiente de Desenvolvimento(Tablespace)

create tablespace tbs\_dados

datafile 'C:\oraclexe\app\oracle\oradata\tbd\_dados.dbf' size 100M reuse

autoextend on next 10M maxsize 200M

online;

#Criar um usuário vinculado ao Tablespace

create user Rafael --usuario

identified by "123" --senha

default tablespace tbs\_dados

temporary tablespace temp;

#Permisionamento

-- Cria e define a "role" de privilegios (perfil)

create role perfil\_desenv;

grant

create cluster,

create database link,

create procedure,

create session,

create sequence,

create synonym,

create table,

create any type,

create trigger,

create view

to perfil\_desenv;

grant alter session to perfil\_desenv;

grant create trigger to rafael;

grant perfil\_desenv to rafael;

grant unlimited tablespace to rafael;

## Seção 1- Criação das Tabelas e Comandos Iniciais

CREATE TABLE TALUNO(

COD\_ALUNO INTEGER NOT NULL,

NOME VARCHAR(30),

CIDADE VARCHAR2(30),

CEP VARCHAR(10),

PRIMARY KEY (COD\_ALUNO)

);

CREATE TABLE TCURSO

( COD\_CURSO INTEGER NOT NULL PRIMARY KEY,

NOME VARCHAR2(30),

VALOR NUMBER(8,2),

CARGA\_HORARIA INTEGER

);

CREATE TABLE TCONTRATO

( COD\_CONTRATO INTEGER NOT NULL PRIMARY KEY,

DATA DATE,

COD\_ALUNO INTEGER,

TOTAL NUMBER(8,2),

DESCONTO NUMBER(5,2)

);

CREATE SEQUENCE SEQ\_ALUNO START WITH 4;

CREATE TABLE TITEM

( COD\_ITEM INTEGER NOT NULL PRIMARY KEY,

COD\_CURSO INTEGER,

COD\_CONTRATO INTEGER,

VALOR NUMBER(8,2) );

## Seção 2 -> Tipos de Dados

INTEGER - 1, 2 -- numero inteiro -> number(38)

NUMBER(5,2) - 999,99

NUMERIC(5,2) - 999,99

DATE - '10/03/2011 00:00:00'

VARCHAR(10) -- Sinonimo

VARCHAR2(10) - 'MARCIO'

CHAR(10) - 'MARCIO ‘

## Seção 4 – Funções Básicas

### Single-Row

Concat -> SELECT Concat(COD\_ALUNO,NOME) FROM TALUNO;

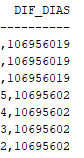
Instr -> SELECT nome, InStr(NOME,'R') FROM TALUNO; //Primeira ocorrência de determinada letra.

Subtr -> SELECT SubStr(NOME,Length(nome)-1, 2) FROM TALUNO;

### Datas

#### #Diferença de dias é considerado também as horas. Fazer o trunc no resultado.

SELECT SYSDATE - DATA AS DIF\_DIAS FROM TCONTRATO;



#### --Somando horas em uma data

SELECT SYSDATE, SYSDATE + 5 / 24 as ADD\_HORAS FROM TCONTRATO;

#### --Somar minutos

SELECT SYSDATE, SYSDATE + 15 / 1440 as ADD\_MINUTOS FROM TCONTRATO;

#### --Somar Segundos

SELECT SYSDATE, SYSDATE + 30 / (3600 \* 24) as ADD\_SEGUNDOS FROM TCONTRATO;

#### --Proxima data a partir de uma dia da semana

SELECT Next\_Day(SYSDATE, 'QUARTA-FEIRA') AS PROXIMA\_QUARTA\_DATA FROM DUAL;

#### --Ultimo dia do mes

SELECT Last\_Day(SYSDATE) AS ULTIMO\_DIA\_MES FROM DUAL;

#### --Primeiro dia do mes

SELECT Trunc(SYSDATE,'MONTH') AS PRIMEIRO\_DIA\_MES\_CORRENTE FROM DUAL;

#### --Primeiro dia do proximo mês

--Até dia 15 do mes pega o primeiro dia do mes atual,--a partir do dia 16 retorna o primeiro dia do proximo mes

SELECT Round(SYSDATE+14, 'MONTH') AS PRIMEIRO\_DIA\_PROXIMO\_MES FROM DUAL;

### ---Formatação de data

#### --Busca o nome do mês

SELECT To\_Char(SYSDATE,'MONTH') MES1 FROM DUAL;

MES1

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

AGOSTO

#### --Busca o nomed do dia do mês

SELECT To\_Char(SYSDATE,'DAY') DIA\_SEMANA1 FROM DUAL;

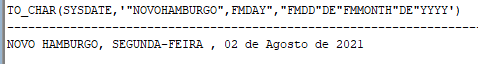
#### --Nome, em extenso, do Ano

SELECT To\_Char(SYSDATE,'YEAR') ANO FROM DUAL;

#### --Concatenar String no To\_char .

o Fm serve para tirar os espaços

SELECT To\_Char(SYSDATE,'"NOVO HAMBURGO", fmDAY "," fmDD "de" fmMonth "de" YYYY') FROM DUAL;



### --Convertendo números no TO\_CHAR

--L -> Coloca o R$

--G -> ponto

--D -> casas decimais

SELECT Salario,Trim(To\_Char(Salario,'L99999.99')) salario1, trim(To\_Char(Salario,'L99G999D99')) salario2 FROM TALUNO;



## Seção 7 – SubConsultas

### Exemplos 1:

SELECT COD\_ALUNO, NOME, CIDADE, ESTADO

FROM TALUNO

WHERE (CIDADE,ESTADO) =

( SELECT CIDADE,ESTADO FROM TALUNO

WHERE COD\_ALUNO = 1 )

AND COD\_ALUNO <> 1 ;

### Exemplo 2:

SELECT COD\_CURSO, Min(VALOR),Sum(VALOR),

Count(\*) QTDE

FROM TITEM

WHERE cod\_curso > 0

GROUP BY COD\_CURSO

HAVING Min(VALOR) >=

(SELECT Avg(VALOR) FROM TCURSO)

ORDER BY Cod\_Curso;

### Exemplo 3:

--SubConsulta na clausula From

SELECT ITE.COD\_CONTRATO, ITE.VALOR, ITE.COD\_CURSO,

CUR.COD\_CURSO codigo, CUR.VALOR

FROM TITEM ITE,

( SELECT COD\_CURSO, VALOR

FROM TCURSO WHERE VALOR > 500 ) CUR

WHERE CUR.COD\_CURSO = ITE.COD\_CURSO

## Seção 8: Variáveis de Substituição e SavePoint

Variáveis de Substituição:

SELECT \* FROM TDESCONTO

WHERE CLASSE = '&cla' and INFERIOR=&inf;

Só executar o comando que aparece uma janela perguntando sobre os dados.

SavePoint -> Ponto de Restauração.

SAVEPOINT upd\_b;

UPDATE TDESCONTO SET

SUPERIOR = 88

WHERE CLASSE = 'B';

--ponto de restauraçao

SAVEPOINT ins\_Ok;

--

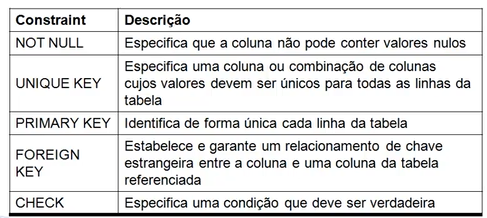
INSERT INTO tdesconto(classe, inferior, superior) VALUES ('&cla', &inf, &sup);

ROLLBACK TO SAVEPOINT ins\_Ok;

ROLLBACK TO SAVEPOINT upd\_a;

ROLLBACK TO SAVEPOINT upd\_b;

## Seção 10 – Constraints:



### --Desabilitando/Habilitando constraint

ALTER TABLE TPESSOA DISABLE CONSTRAINT uk\_cpf;

ALTER TABLE TPESSOA ENABLE CONSTRAINT uk\_cpf;

### --Excluir Constraint

ALTER TABLE TPESSOA DROP CONSTRAINT uk\_cpf;

## Seção 11 – Views (Visões)

É um SELECT em formato de objeto.

Existem 2 tipos:

Views Simples -> Uma tabela, Não Contém Função, Não Possui Grupos de Dados, DML atráves de visões.

Views Complexas -> Uma ou mais tabelas. Contém Função. Possui grupos de Dados. Pode executar DML.

### Ex:

CREATE OR REPLACE VIEW V\_ALUNO

AS

SELECT COD\_ALUNO AS CODIGO, SALARIO, ESTADO,

NOME AS ALUNO, CIDADE

FROM TALUNO

WHERE ESTADO='RS';

### --View com parametros de saida

CREATE OR REPLACE VIEW V\_ALUNO2(COD, ALUNO, SAL)

AS

SELECT COD\_ALUNO, NOME, SALARIO

FROM TALUNO;

### --View Complexa

**--(nao pode fazer DML em view complexa)**

CREATE OR REPLACE VIEW V\_CONTRATO

AS

SELECT Trunc(DATA) AS DATA,

Max(DESCONTO) MAXIMO,

Avg(DESCONTO) MEDIA,

Count(\*) QTDE

FROM TCONTRATO

GROUP BY Trunc(DATA);

### --OPERACAO DML NA VIEW

CREATE OR REPLACE VIEW vcursos1000ck

AS

SELECT cod\_curso, nome, valor

FROM TCurso

WHERE VALOR = 1000

**WITH CHECK OPTION CONSTRAINT vcursos1000\_ck; --Isso serve para ter um critério de Insert. Só insere caso o VALOR seja igual a 1000.**

INSERT INTO vCursos1000ck

(cod\_curso, nome, valor)

VALUES (52,'TESTE Y', 1000);

### --View somente leitura (Nao permite DML)

CREATE OR REPLACE VIEW V\_ALUNO3

AS

SELECT COD\_ALUNO CODIGO,

NOME ALUNO, CIDADE

FROM TALUNO

WHERE ESTADO='RS'

**WITH READ ONLY**;

## Seção 13 - Sequence

CREATE SEQUENCE SEQ\_ALUNO1

START WITH 60

--Valor Inicial

INCREMENT BY 2

--Qtde a Incrementar

MINVALUE 60

--Valor Minimo

MAXVALUE 100

--Valor Maximo

NOCACHE –Não guarda o Cache. Por padrão é 20.

--nao guarda em cache faixa de valores -- 20

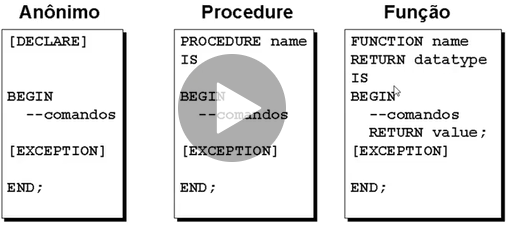
NOCYCLE; -- No Cycle serve para, caso chegue no máximo da sequence, ele reinicia o contador.

#### --Valor Atual

SELECT SEQ\_ALUNO1.CURRVAL FROM DUAL;

## PL/SQL

### 1 – Corpo de um bloco Anônimo, Procedure e Função.



#### Exemplo Procedure:

CREATE OR REPLACE PROCEDURE P\_AUMENTA\_PRECOCURSO

IS

--variaveis

BEGIN

UPDATE TCURSO SET VALOR = VALOR \* 1.20;

END;

--Executa procedure 1

EXECUTE P\_AUMENTA\_PRECOCURSO;

--Executa procedure 2 - Bloco anonimo

BEGIN

P\_AUMENTA\_PRECOCURSO;

END;

#### Declarando variáveis:

vData Date;

vPreco Number(2) NOT NULL :=10;

vCidade varchar2(60) := ‘Novo Hamburgo’;

cDesconto NUMBER := 100;

#### Tipos de Variáveis:

Long -> Dados de Caractéres do tamanho variável de até 2GB.

Long Row -> Dados Binários e strings do tipo byte de até 32760 bytes.

Não são interpretados pelo PL/SQL.

Binary\_float -> Ponto flutuante com precisão simples.

Binary\_double -> Ponto flutuante com precisão dupla.

Binary\_integer -> Tipo básico para inteiro.

PLS\_INTEGER -> Tipo básico para inteiro com sinal.

São mais rápidos do que NUMBER e BINARY\_INTEGER.

Atributo %Type -> Utilizar o tipo da variável de uma tabela.

vNome tcliente.nome%TYPE;

Boolean -> vFlag BOOLEAN := (vPreco < vValor)

Record -> Similar a um Array.

Collection ->

Lob ->

CLOB = Texto

BLOB = Foto, Word

BFILE = Filme

NCLOB = NCLOB

Variáveis não PL/SQL -> São mais performáticos, a partir da segunda execução.

:gPReco\_final := vPreco \* 0.85;

### 2 – Exemplo utilizando o Dbms\_Output.Put\_Line

SET SERVEROUTPUT ON

DECLARE

VDESCONTO NUMBER(6,2) := 0.50;

VCIDADE VARCHAR(30) := 'NOVO HAMBURGO';

VCOD\_ALUNO TALUNO.COD\_ALUNO%TYPE := 5;

VTOTAL NUMBER(8,2) := 1345.89;

vFlag BOOLEAN;

BEGIN

VTOTAL := Round(VTOTAL \* VDESCONTO, 2);

vFlag := (VCOD\_ALUNO>6);

IF (vFlag=True) THEN --Se vFlag = True Entao

Dbms\_Output.Put\_Line('Verdadeiro');

ELSE --Senao

Dbms\_Output.Put\_Line('Falso');

END IF;

Dbms\_Output.Put\_Line('Total: '|| vTotal);

VDESCONTO := 1.20;

vCIDADE := InitCap(VCIDADE);

Dbms\_Output.Put\_Line('Cidade: '||vCidade);

Dbms\_Output.Put\_Line('Desconto: '||VDESCONTO);

Dbms\_Output.Put\_Line('Aluno: '||VCOD\_ALUNO);

END;

#### Bind variable

--Variável fora do Contexto. É mais rápido

-- Utilizar o “:”. Isso ocorre porque ele identifica que não precisa recompilar novamente esta variável.

VARIABLE vDESCONTO2 NUMBER

DECLARE

VCOD\_ALUNO NUMBER := 1;

BEGIN

:vDESCONTO2 := 0.90;

Dbms\_Output.put\_line('desconto 2: '||:vDESCONTO2);

UPDATE TContrato SET

TOTAL = TOTAL \* :vDESCONTO2

WHERE COD\_ALUNO = VCOD\_ALUNO;

END;

### 3 – Quantidade de Linhas Afetadas:

SET SERVEROUTPUT ON

DECLARE

vContrato NUMBER := &cod\_contrato;

vtexto VARCHAR2(50);

BEGIN

UPDATE TContrato SET

desconto = desconto + 2

WHERE Cod\_Contrato = VContrato;

**vtexto := SQL%ROWCOUNT**;

--Retorna qtde de registros afetados

--pelo comando sql

Dbms\_Output.Put\_Line(vtexto|| ' linhas atualizadas.');

END;

### 4 – Loops

#### Loop

--Laço de repetição 1 até 10

DECLARE

vContador INTEGER := 0;

BEGIN

**LOOP**

vContador := vContador + 1;

Dbms\_Output.Put\_Line(vContador);

EXIT WHEN vContador = 10;

**END LOOP**;

Dbms\_Output.Put\_Line('Fim do LOOP');

END;

#### For Loop

--For loop -> mais indicado para laços em tabelas

DECLARE

vContador INTEGER;

**BEGIN**

**FOR vContador in 1..10**

**LOOP**

--vContador := vContador + 1;

Dbms\_Output.Put\_Line(vContador);

--EXIT WHEN vContador = 5;

**END LOOP;**

END;

#### While Loop

DECLARE

vContador INTEGER := 0;

vTexto VARCHAR(10);

BEGIN

**WHILE vContador < 10**

**LOOP**

vContador := vContador + 1;

IF (vContador Mod 2)=0 THEN

vTexto := 'Par';

ELSE

vTexto := 'Impar';

END IF;

Dbms\_Output.Put\_Line(vContador|| ' -> '||vTexto);

**END LOOP;**

END;

### 5 – Records

Grupo de Itens de Dados.

Como se fosse uma tabela.

#### Ex 1:

DECLARE

TYPE Rec\_aluno IS RECORD

( cod\_aluno NUMBER NOT NULL := 0,

nome TALUNO.Nome%TYPE,

cidade TALUNO.Cidade%TYPE );

Registro rec\_aluno;

BEGIN

registro.cod\_aluno := 50;

registro.nome := 'Master';

registro.cidade := 'Novo Hamburgo';

Dbms\_Output.Put\_Line('Codigo: '||registro.cod\_aluno);

Dbms\_Output.Put\_Line(' Nome: '||registro.nome);

Dbms\_Output.Put\_Line('Cidade: '||registro.cidade);

END;

#### Ex 2:

DECLARE

reg TAluno%ROWTYPE; --Record

vcep VARCHAR(10) := '98300000';

BEGIN

SELECT COD\_ALUNO, NOME, CIDADE

INTO Reg.cod\_aluno, Reg.nome, Reg.cidade

FROM TALUNO

WHERE COD\_ALUNO = 1;

vCep := '93500000';

reg.cep := vCep;

Dbms\_Output.Put\_Line('Codigo: ' ||reg.cod\_aluno);

Dbms\_Output.Put\_Line('Nome : ' ||reg.nome);

Dbms\_Output.Put\_Line('Cidade: ' ||reg.cidade);

Dbms\_Output.Put\_Line('Cep : ' ||reg.cep);

END;

### 6 – Collection



DECLARE

TYPE nome\_type IS TABLE OF taluno.nome%TYPE;

nome\_table nome\_type := nome\_type(); --Criando Instancia

BEGIN

nome\_table.EXTEND; -- alocando uma nova linha

nome\_table(1) := 'Marcelo';

nome\_table.EXTEND; -- alocando uma nova linha

nome\_table(2) := 'Marcio';

Dbms\_Output.Put\_Line('Nome 1: '||nome\_table(1));

Dbms\_Output.Put\_Line('Nome 2: '||nome\_table(2));

END;

Gerenciamento de múltiplas linhas de dados. Ex: Vetores

Tipos:

#### Associative Arrays (Index-by-tables)-> Tamanho variável

Não necessita da utilização do método EXTEND para alocação de novos elementos.

Pode ser indexado com qualquer valor.

Obrigatória a cláusula INDEX BY

##### Ex 1:

DECLARE

TYPE T\_ALUNO IS TABLE OF TALUNO.NOME%TYPE

INDEX BY BINARY\_INTEGER; --Matriz

REGISTRO T\_ALUNO; --Record

teste INTEGER;

tetste2 INTEGER;

BEGIN

REGISTRO(1) := 'MARCIO';

REGISTRO(2) := 'JOSE';

REGISTRO(3) := 'PEDRO';

teste:=REGISTRO.count();

FOR tetste2 IN 1..REGISTRO.count()

LOOP

Dbms\_Output.Put\_Line('Nome 1: '||registro(tetste2));

-- Dbms\_Output.Put\_Line('Nome 2: '||registro(2));

-- Dbms\_Output.Put\_Line('Nome 3: '||registro(3));

END LOOP;

END;

#### Nested Tables -> Tamanho variável

##### Ex 1:

DECLARE

TYPE T\_ALUNO IS TABLE OF TALUNO%ROWTYPE

INDEX BY BINARY\_INTEGER; --Matriz

C NUMBER;

REGISTRO T\_ALUNO; --Record

BEGIN

C:=1;

for cada\_linha in (Select \* from TALUNO) LOOP

REGISTRO(C).COD\_ALUNO := cada\_linha.COD\_ALUNO;

REGISTRO(C).NOME := cada\_linha.NOME;

REGISTRO(C).CIDADE := cada\_linha.CIDADE;

REGISTRO(C).CEP := cada\_linha.CEP;

REGISTRO(C).ESTADO := cada\_linha.ESTADO;

REGISTRO(C).SALARIO := cada\_linha.SALARIO;

REGISTRO(C).NASCIMENTO := cada\_linha.NASCIMENTO;

C:=C+1;

END LOOP;

FOR cada\_linha IN 1..REGISTRO.Count()

LOOP

Dbms\_Output.Put\_Line('Cod\_Aluno : '||registro(cada\_linha).COD\_ALUNO);

Dbms\_Output.Put\_Line('Nome : '||registro(cada\_linha).NOME);

END LOOP;

END;

##### Ex 2:

DECLARE

TYPE tipo IS TABLE OF VARCHAR2(30) INDEX BY VARCHAR2(2);

--

uf\_capital tipo;

BEGIN

uf\_capital('RS') := 'PORTO ALEGRE';

uf\_capital('RJ') := 'RIO DE JANEIRO';

uf\_capital('PR') := 'CURITIBA';

uf\_capital('MT') := 'CUIABA';

dbms\_output.put\_line(uf\_capital('RS'));

dbms\_output.put\_line(uf\_capital('RJ'));

dbms\_output.put\_line(uf\_capital('PR'));

dbms\_output.put\_line(uf\_capital('MT'));

END;

#### Varrays -> Semelhante a matriz.

DECLARE

TYPE nome\_varray IS VARRAY(5) OF taluno.nome%TYPE;

nome\_vetor nome\_varray := nome\_varray();

BEGIN

nome\_vetor.EXTEND;

nome\_vetor(1) := 'Master';

Dbms\_Output.Put\_Line(nome\_vetor(1));

END;

### 7 – Cursores Explícitos

Declare -> Open -> Fetch -> Vaz

%ISOPEN -> Retorna se o Cursor estiver aberto

%NOTFOUND -> Retorna TRUE se o fetch mais recente não retornou uma linha.

%FOUND -> Contrário do NOTFOUND

%ROWCOUNT -> Retorna o número total de linhas recuperadas até o momento.

##### Exemp 1 – Cursor para recuperar campos:

DECLARE

vcod\_aluno TAluno.Cod\_Aluno%TYPE;

vNome TAluno.nome%TYPE;

CURSOR c1 IS

SELECT cod\_aluno, nome

FROM taluno;

BEGIN

OPEN c1; -- abre cursor

LOOP

FETCH c1 INTO vCod\_Aluno, vNome; --pega registro atual

EXIT WHEN c1%ROWCOUNT > 10 OR c1%NOTFOUND;

Dbms\_Output.Put\_Line('Codigo: '||

LPad(vcod\_aluno,4,'0')||' - '||'Nome: '||vNome);

END LOOP;

CLOSE c1; --fecha cursor

END;

##### Exemp 2 – Cursor para recuperar vários campos:

DECLARE

CURSOR c1 IS

SELECT \* FROM TAluno;

Reg c1%ROWTYPE; --record

BEGIN

OPEN c1; --

LOOP

FETCH c1 INTO reg;

EXIT WHEN c1%ROWCOUNT > 10 OR c1%NOTFOUND;

Dbms\_Output.Put\_Line('Codigo: '||

LPad(reg.cod\_aluno,5,'0')||'-'||

'Nome: '||reg.nome);

END LOOP;

CLOSE c1; --

END;

##### Exemp 3- Cursos passando parâmetro

DECLARE

CURSOR c1 (pCod\_aluno NUMBER) IS

SELECT \* FROM TAluno

WHERE Cod\_aluno = pCod\_aluno

FOR UPDATE OF NOME NOWAIT;--bloquia

--bloquea a coluna nome para alteracao

Reg c1%ROWTYPE;

BEGIN

OPEN c1(&codigo);

FETCH c1 INTO reg;

Dbms\_Output.Put\_Line(reg.cod\_aluno ||' - ' || reg.nome);

CLOSE c1; --libera o registro para alteracao

END;

##### Exemp 4- Cursos travando tabela

DECLARE

CURSOR c1 IS

SELECT \* FROM TALUNO

FOR UPDATE;

Reg\_aluno c1%ROWTYPE;

BEGIN

FOR reg\_aluno IN c1

LOOP

UPDATE TALUNO

SET nome = InitCap(reg\_aluno.nome)

WHERE CURRENT OF c1; --bloqueia somente o reg atual

Dbms\_Output.Put\_Line('Nome: '||InitCap(reg\_aluno.nome));

END LOOP;

COMMIT;

END;

### 8 – Tratamento de Erros (Exceções):

##### Exemplo 1 – Exceções por padrão:

DECLARE

vCod taluno.cod\_aluno%TYPE := 566;

vCidade taluno.cidade%TYPE; x NUMBER;

BEGIN

SELECT Cidade INTO vCidade

FROM TAluno

WHERE nome LIKE '%';

--WHERE cod\_aluno = vCod;

X := 0 / 0;

Dbms\_Output.Put\_Line('Cidade: '||vCidade);

EXCEPTION

WHEN no\_data\_found THEN

RAISE\_APPLICATION\_ERROR(-20001,

'Aluno Inexistente! '||SQLCODE||' '||SQLERRM);

WHEN too\_many\_rows THEN

RAISE\_APPLICATION\_ERROR(-20002,

'Registro Duplicado! '||SQLCODE||' '||SQLERRM);

WHEN others THEN

RAISE\_APPLICATION\_ERROR(-20003,

'Exceção Desconhecida! '||SQLCODE||' '||SQLERRM);

END;

##### Exemplo 2 – Exceções desenvolvidas manualmente

DECLARE

vDt\_vencimento DATE;

vConta NUMBER := 100; --codigo da conta

**eConta\_vencida EXCEPTION;**

BEGIN

SELECT vencimento INTO vDt\_vencimento

FROM CONTAS WHERE codigo = vConta;

IF vDt\_vencimento < TRUNC(SYSDATE) THEN

RAISE eConta\_vencida;

END IF;

EXCEPTION

**WHEN eConta\_vencida THEN**

Dbms\_Output.Put\_Line('Conta vencida');

UPDATE contas SET valor = valor + juros

WHERE codigo = vConta;

END;

### 9 – Procedures

Parâmetros devem ser declarados como um tipo de dado escalar (varchar2, number, etc..) sem precisão ou com um tipo %TYPE ou %RowType.

3 Tipos de Parâmtros:

IN

OUT

IN OUT

#### Exempl 1:

CREATE OR REPLACE PROCEDURE aumenta\_precocurso(pCod\_Curso NUMBER)

IS

-- Variaveis

BEGIN

--Aumenta 50%

UPDATE TCURSO SET

VALOR = VALOR \* 1.5

WHERE COD\_CURSO = pCod\_Curso;

END;

--Teste

SELECT \* FROM TCURSO;

EXEC AUMENTA\_PRECOCURSO(1); --Executa a procedure

--com bloco anonimo

declare

vcod INTEGER := &codigo;

begin

AUMENTA\_PRECOCURSO(vcod);

end;

#### Exemplo 2:

CREATE OR REPLACE PROCEDURE Consulta\_Aluno

( pCodigo IN TALUNO.Cod\_Aluno%TYPE,

pNome OUT TALUNO.Nome%TYPE,

pCep OUT TALUNO.Cep%TYPE,

pCidade OUT TALUNO.Cidade%TYPE)

IS

BEGIN

SELECT nome, Cep, cidade

INTO pNome, pCep, pCidade

FROM taluno

WHERE cod\_aluno = pCodigo;

END;

--Teste

DECLARE

vnome VARCHAR2(30);

vcidade VARCHAR(30);

vcep VARCHAR(10);

vCod INTEGER := &Codigo;

BEGIN

--executa a procedure

-- Consulta\_aluno(vcod, vnome, vcep, vcidade);

Consulta\_Aluno(pNome => vnome,

pCodigo => vcod,

pCidade => vcidade,

pCep => vcep);

Dbms\_Output.Put\_Line(vNome);

Dbms\_Output.Put\_Line(vCidade);

Dbms\_Output.Put\_Line(vcep);

END;

#### Exemplo 3

CREATE OR REPLACE PROCEDURE Consulta\_Aluno2

(pRegistro IN OUT TALUNO%ROWTYPE)

IS

CURSOR CSQL IS

SELECT \* FROM TALUNO WHERE cod\_aluno = pRegistro.COD\_ALUNO;

BEGIN

OPEN CSQL;

FETCH CSQL INTO pRegistro;

CLOSE CSQL;

END;

--Teste

DECLARE

vRegistro TALUNO%ROWTYPE;

BEGIN

vRegistro.COD\_ALUNO := &Codigo;

Consulta\_Aluno2(vRegistro);

Dbms\_Output.Put\_Line(vRegistro.Nome);

Dbms\_Output.Put\_Line(vRegistro.Cidade||' Cep: '||vRegistro.Cep);

END;

### 10) Functions

#### Exemplo 1 – função básica

CREATE OR REPLACE FUNCTION CONSULTA\_PRECO

(pCod\_Curso NUMBER) RETURN NUMBER

AS

vValor NUMBER;

BEGIN

SELECT valor INTO vValor FROM TCurso

WHERE cod\_curso = pCod\_Curso;

RETURN(vValor);

END;

/

--Teste | Usando function

DECLARE

vCod NUMBER := &codigo;

vValor NUMBER;

BEGIN

vValor := consulta\_preco(vCod);

Dbms\_Output.Put\_Line('Preco do curso: '||vValor);

END;

#### Exemplo 2 – Pipelined Table Function

Permite criar funções que retornam dados como se fossem uma tabela virtual

A instrução PIPE ROW retorna os resultados para a sessão de usuário Oracle, linha por linha. Isso otimiza o tempo de resposta da aplicação.

CREATE OR REPLACE TYPE REG\_ALUNO AS OBJECT

( CODIGO INTEGER,

NOME VARCHAR2(30),

CIDADE VARCHAR(30) );

--Matriz

CREATE OR REPLACE TYPE TABLE\_REG\_ALUNO AS TABLE OF REG\_ALUNO;

--Function que retorna registros

CREATE OR REPLACE FUNCTION GET\_ALUNO(pCODIGO NUMBER)

RETURN TABLE\_REG\_ALUNO PIPELINED

IS

outLista REG\_ALUNO;

CURSOR CSQL IS

SELECT ALU.COD\_ALUNO, ALU.NOME, ALU.CIDADE

FROM TALUNO ALU

WHERE ALU.COD\_ALUNO = pCODIGO;

REG CSQL%ROWTYPE;

BEGIN

OPEN CSQL;

FETCH CSQL INTO REG;

outLista := REG\_ALUNO(REG.COD\_ALUNO, REG.NOME, REG.CIDADE);

PIPE ROW(outLista); --Escreve a linha

CLOSE CSQL;

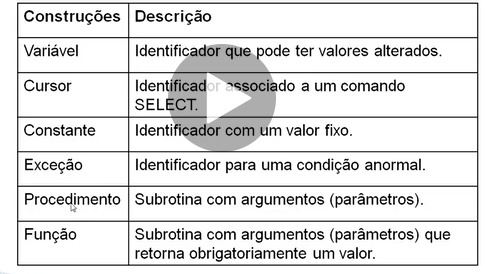
RETURN;

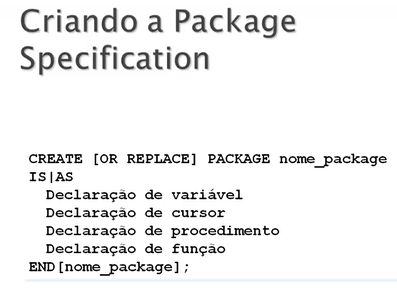
END;

--usando

SELECT \* FROM TABLE(GET\_ALUNO(1));

### 11 – Packages





Serve para agrupar objetos PL/SQL

Crie uma Package em duas partes:

- Package Specification

-Package Body

Compilando uma Package

Alter Package nome\_package COMPILE;

#### Exemplo 1:

CREATE OR REPLACE PACKAGE PKG\_ALUNO

IS

vCIDADE VARCHAR2(30); --Variaveis publicas

vMedia NUMBER(8,2); --Variaveis publicas

vNOME VARCHAR2(30); --Variaveis publicas

PROCEDURE DELETA\_ALUNO(pCOD\_ALUNO NUMBER);

PROCEDURE MEDIA\_CONTRATOS;

PROCEDURE CON\_ALUNO(pCOD\_ALUNO NUMBER);

END;

PROCEDURE MEDIA\_CONTRATOS

IS

BEGIN

vTESTE := 'teste';

SELECT Avg(total) INTO vMEDIA FROM tcontrato;

END;

--\*\*\*\*\*\*\*\*\*\*\*\*

PROCEDURE CON\_ALUNO(pCOD\_ALUNO NUMBER)

IS

BEGIN

vNOME := '';

SELECT NOME INTO vNOME FROM TALUNO

WHERE COD\_ALUNO=pCOD\_ALUNO;

EXCEPTION

WHEN No\_Data\_Found THEN

Dbms\_Output.Put\_Line('Aluno não existe');

END;

---\*\*\*\*\*\*\*\*\*\*\*\*\*

PROCEDURE DELETA\_ALUNO(pCOD\_ALUNO NUMBER)

IS

BEGIN

CON\_ALUNO(pCOD\_ALUNO);

IF Length(vNOME) > 0 THEN

DELETE FROM TALUNO WHERE COD\_ALUNO = pCOD\_ALUNO;

Dbms\_Output.Put\_Line(vNOME||'->Excluido');

END IF;

END;

END;

--USANDO

EXEC PKG\_ALUNO.DELETA\_ALUNO(666);

DECLARE

m NUMBER;

BEGIN

pkg\_aluno.media\_contratos; --executa a procedure

m := pkg\_aluno.vMedia;

Dbms\_Output.Put\_Line('Média: '||m);

pkg\_aluno.con\_aluno(89); --executa a procedure

nome := pkg\_aluno.vnome;

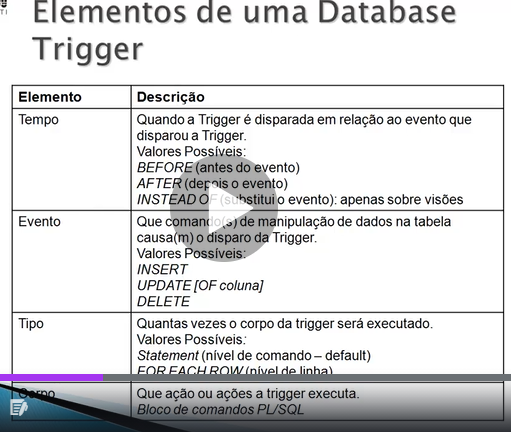
Dbms\_Output.Put\_Line('Nome '||nome);

END;

### 12 - Triggers

6 tipos de Triggers:

Trigger a Nível de Comando vs Nível de Linha.



#### Exemplo 1 – Trigger Simples. Nível de Comando

CREATE OR REPLACE TRIGGER VALIDA\_HORARIO\_CURSO

BEFORE INSERT OR DELETE ON TContrato

BEGIN

IF(TO\_CHAR(SYSDATE, 'D') IN (1) OR

To\_Number(To\_Char(SYSDATE,'HH24'))

NOT BETWEEN 8 AND 18) THEN

Raise\_Application\_Error(-20001,'Fora horário comercial');

END IF;

END;

--

INSERT INTO TCONTRATO

VALUES (7665, SYSDATE, 10, 1500, NULL);

#### Exemplo 2 – Trigger com condição específica de um campo. Nível de Comando

CREATE OR REPLACE TRIGGER valida\_horario\_curso2

BEFORE INSERT OR UPDATE OR DELETE ON TCONTRATO

BEGIN

IF(TO\_CHAR(SYSDATE,'D') IN (1, 7) OR

TO\_NUMBER(To\_Char(SYSDATE,'HH24'))NOT BETWEEN 8 AND 18)THEN

IF( INSERTING ) THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Não pode inserir');

ELSIF( DELETING ) THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Não pode remover');

ELSIF( UPDATING('TOTAL') ) THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Não pode alterar total');

ELSE

RAISE\_APPLICATION\_ERROR(-20004, 'Não pode alterar');

END IF;

END IF;

END;

#### Exemplo 3 – Trigger Nível de linha para determinado campo.

CREATE OR REPLACE TRIGGER gera\_Log\_CURSO

BEFORE UPDATE OF VALOR ON TCURSO

FOR EACH ROW

BEGIN

INSERT INTO Log( Usuario, data, obs,

Valor\_antigo, Valor\_novo)

VALUES ( USER, SYSDATE,'Curso Alterado: '||:OLD.NOME,

:OLD.VALOR, :NEW.VALOR );

END;

#### Exemplo 4 – Trigger nível de Linha para toda a Tabela.

CREATE OR REPLACE TRIGGER tri\_atual\_prereq

AFTER UPDATE ON tcurso FOR EACH ROW

BEGIN

UPDATE tcurso

SET pre\_req = :new.cod\_curso

WHERE pre\_req = :old.cod\_curso;

END;

#### Exemplo 5 – Trigger INSTEAD OF

Usada para as Views.

CREATE OR REPLACE VIEW vcontratos\_pares

AS SELECT COD\_CONTRATO, DATA, COD\_ALUNO, DESCONTO, TOTAL

FROM tcontrato;

CREATE OR REPLACE TRIGGER tri\_contratos\_pares

INSTEAD OF INSERT OR DELETE OR UPDATE ON vcontratos\_pares

DECLARE

BEGIN

INSERT INTO Log( usuario, data, obs )

VALUES (USER, SYSDATE, 'DML da view VCONTRATOS\_PARES.' );

END;

### 13 – Avançado Group by – RollUp, Cube. Rank, Dense\_Rank\_Ration\_to\_Report, LED, LAG

São subtotais e tabulações sobre dimensões.

Uteis para DataWareHouse (Apresentar os Subtotais)

ROLLUP, retorna uma linha sumarizada para os subgrupos.

CUBE ->

Grouping -> Retorna 1 se o valor da exmpressão retornar um SubGrupo

Grouping\_id

Rank() -> Posição, Ordem das linhas, começa com 1. Ranqueamento

Rank() com Partition By

Dense\_Rank -> Não empata os valores iguais no Rank

Ration\_To\_Report -> Calcula a proporção de um valor em relação à agregação de um conjunto de valores.

#### Ex 1: Usando RollUp no Group By – Serve para criar as subcategorias

--

SELECT COD\_ALUNO,

CASE

WHEN TRUNC(DATA) IS NULL AND COD\_ALUNO IS NOT NULL

THEN 'SUB-TOTAL'

WHEN COD\_ALUNO IS NULL

THEN 'TOTAL-GERAL'

ELSE TO\_CHAR( Trunc(DATA) )

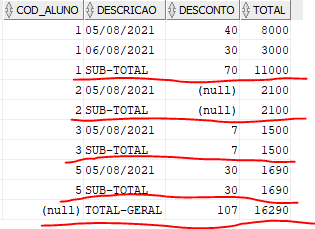
END DESCRICAO,

sum(DESCONTO) DESCONTO,

SUM(TOTAL) TOTAL

FROM TCONTRATO

GROUP BY ROLLUP(COD\_ALUNO, TRUNC(DATA) );

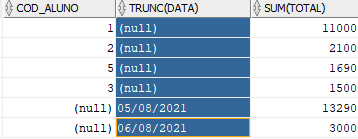


#### Ex 2: Group By Grouping Sets -> Retorna apenas os SubTotais

SELECT COD\_ALUNO, Trunc(DATA), SUM(TOTAL)

FROM TCONTRATO

GROUP BY GROUPING SETS (COD\_ALUNO, Trunc(DATA) );



#### Ex 3: Rank () -> Usar o Rankeamento de posição. Primeiro, Segundo , Terceiro

SELECT Trunc(DATA),

COD\_ALUNO,

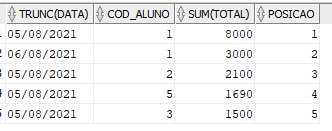
SUM(TOTAL),

RANK() OVER (ORDER BY SUM(TOTAL) DESC) POSICAO --Aqui, a primeira posição será a do retorno do Select.

--Como está DESC, a primeira do Rank será a de maior valor Total

FROM TCONTRATO

GROUP BY (Trunc(DATA), COD\_ALUNO)



#### Ex 4: Rank com PARTITION BY -> Faz o Ranqueamento considerando determinado Campo

SELECT Trunc(DATA),

COD\_ALUNO,

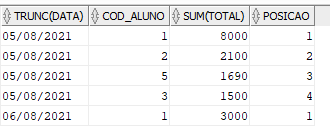
SUM(TOTAL),

RANK() OVER (PARTITION BY Trunc(DATA) ORDER BY SUM(TOTAL) DESC) POSICAO

FROM TCONTRATO

GROUP BY (Trunc(DATA),COD\_ALUNO)

ORDER BY Trunc(DATA);



#### Ex 5: Dense Rankin -> Não deixa empatado

SELECT COD\_ALUNO, TOTAL,

RANK() OVER (ORDER BY TOTAL DESC) "RANK()",

DENSE\_RANK() OVER (ORDER BY TOTAL DESC) "DENSE\_RANK()"

FROM TCONTRATO

GROUP BY COD\_ALUNO, TOTAL;



#### Ex 6: RATIO\_TO\_REPORT -> Serve para mostrar o Percentual

select COD\_ALUNO,Total

,ROUND(RATIO\_TO\_REPORT(TOTAL) OVER()\*200 ,2)"% do Total"

from

(SELECT COD\_ALUNO,

SUM(TOTAL) Total

--ROUND(RATIO\_TO\_REPORT(SUM(TOTAL)) OVER() ,2)"% do Total"

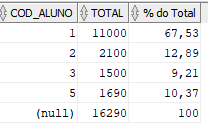
FROM TCONTRATO

--GROUP BY COD\_ALUNO

GROUP BY ROLLUP(COD\_ALUNO))

GROUP BY COD\_ALUNO,Total

order by COD\_ALUNO ;



#### Ex 7: LAG e LEAD –

SELECT Trunc(DATA), SUM(Total) total\_dia,

LAG (SUM(Total),1) OVER (ORDER BY Trunc(DATA)) anterior,

LEAD (SUM(Total),1) OVER (ORDER BY Trunc(DATA)) proximo

FROM TCONTRATO

GROUP BY Trunc(DATA)

ORDER BY Trunc(DATA);



### 14 – Insert All e Merge

INSERT ALL

WHEN TOTAL>=1000 THEN

INTO TCURSO (COD\_CURSO, NOME, VALOR)

VALUES (seq\_curso.NEXTVAL, 'CURSO>1000', TOTAL)

WHEN DESCONTO IS NULL THEN

INTO TCURSO (COD\_CURSO, NOME, VALOR)

VALUES (seq\_curso.NEXTVAL, 'DESCONTO IS NULL', TOTAL)

SELECT COD\_CONTRATO, TOTAL, DESCONTO

FROM TCONTRATO WHERE COD\_CONTRATO = 1;

MERGE INTO TCONTRATO tcn

USING (SELECT COD\_ALUNO AS ALUNO

FROM TALUNO

WHERE estado = 'RS')

ON (tcn.COD\_ALUNO = ALUNO)

WHEN MATCHED THEN --Encontrou o registro

UPDATE SET desconto = 22

WHEN NOT MATCHED THEN --nao encontrou o registro

INSERT(tcn.COD\_CONTRATO, tcn.DATA, tcn.COD\_ALUNO,

tcn.desconto, tcn.total)

VALUES( Seq\_Con.NextVal, SYSDATE, ALUNO, 0, 666);

### 15- Sql Dinâmico

Sql dentro de uma string

CREATE OR REPLACE PROCEDURE consulta\_generica

(pColunas IN VARCHAR2,

pTabelas IN VARCHAR2,

pCondicoes IN VARCHAR2 )

IS

TYPE refCursor IS REF CURSOR;

cCursor1 refCursor;

vRetorno VARCHAR2(4000);

vRet VARCHAR2(4000);

BEGIN

OPEN cCursor1 FOR ' SELECT '||pColunas||

' FROM '||pTabelas||

' WHERE '||pCondicoes;

LOOP

FETCH cCursor1 INTO vRetorno, vret;

EXIT WHEN cCursor1%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(vRetorno||' - '||vret);

END LOOP;

CLOSE cCursor1;

END;

--

SET SERVEROUTPUT ON

EXEC Consulta\_Generica ('NOME,CIDADE','TALUNO','1=1');

# PL/SQL AVANÇADO

## 1 Permissionamento para Debugar.

select \* from user\_sys\_privs

grant debug any procedure to Rafael;

grant debug connect session to Rafael;

## 2 Record e Collections

### Record -> É como se fosse um Array.

Type type\_name is RECORD (idade number(2),.....);

Identifier type\_name;

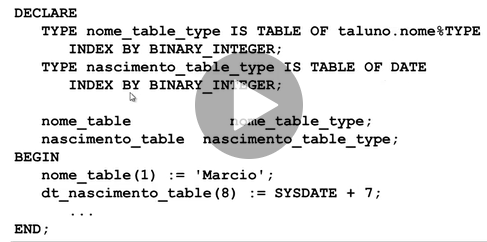
### Collections -> Gerenciamento de múltiplas linhas de dados

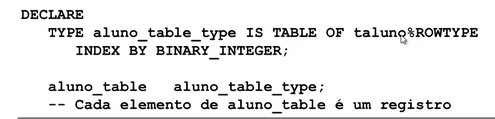
Associative Arrays -> Não precisa do método EXTEND.

Type type\_name is TABLE OF (idade number,.....)

INDEX BY BINARY\_INTEGER | VARCHAR2(20)/

Identifier type\_name





Nested Tables -> Os elementos precisam ser alocados com o EXTEND.

#### VARRAYS

DECLARE

TYPE nome\_varray IS VARRAY(5) OF taluno.nome%TYPE;

nome\_vetor nome\_varray := nome\_varray();

BEGIN

nome\_vetor.EXTEND;

nome\_vetor(1) := 'MasterTraining';

Dbms\_Output.Put\_Line(nome\_vetor(1));

END;

## 3 – Query LEVEL para gerar uma lista

SELECT LEVEL

--, DBMS\_RANDOM.STRING('A', 40)

FROM DUAL CONNECT BY LEVEL <= 100;

## 4 – Full Text Search

Indexar, pesquisar e analisar textos e documentos binários armazenados nas tabelas

--Grant

grant create any directory to Rafael

--Criando um diretório

create or replace directory ARQUIVOS as 'C:\Temp';

--Criando a tabela

create table teste (

codigo number,

nome varchar2(40),

documento blob

);

create sequence seq\_doc;

-- Proceudre para carregar o arquivo para o banco de dados

create or replace procedure grava\_arquivo (p\_file in varchar2)

as

v\_bfile bfile;

v\_blob blob;

begin

insert into teste (codigo,nome,documento)

values (seq\_doc.nextval,p\_file,empty\_blob())

return documento into v\_blob;

-- Informação de directory tem que ser maiusculo

v\_bfile := bfilename('ARQUIVOS',p\_file);

dbms\_lob.fileopen(v\_bfile, dbms\_lob.file\_readonly);

dbms\_lob.loadfromfile(v\_blob,v\_bfile,dbms\_lob.getlength(v\_bfile));

dbms\_lob.fileclose(v\_bfile);

commit;

end;

--Grava o Arquivo

execute grava\_arquivo('arquivo.doc');

Select \* from teste;

Select dbms\_lob.getlength(documento) bytes from teste;

--Vamos criar índice que vai permitir pesquisar dentro deste arquivo grava na tabela

create index ind\_teste\_doc on teste (documento) indextype is ctxsys.context parameters ('sync (on commit)');

--Para verificar se houve erro na criação do índice

select \* from ctx\_user\_index\_errors;

--Podemos verificar que foram criados alguns índices adicionais usando o selects abaixo

select table\_name from user\_tables;

select index\_name,table\_name from user\_indexes;

--Fazendo pesquisar no documento gravando na tabela

select codigo, nome from teste where contains(documento, 'Rafael', 1) > 0;

## 5 – Estatísticas

Coletando estatísticas para o otimizador de queries do Oracle

Estatísticas - Scripts

--Analisa apenas uma tabela - executar como usuário normal

ANALYZE TABLE TALUNO COMPUTE STATISTICS;

--Estatística de schema –

SYSTEM EXEC DBMS\_UTILITY.ANALYZE\_SCHEMA('CURSO','COMPUTE');

--Estatística de banco inteiro (Pode ser demorado) –

SYSTEM EXEC DBMS\_STATS.GATHER\_DATABASE\_STATS;

--Bloco anonimo para ler estatísticas do banco de dados - SYSTEM

1. ECLARE
2. libcac NUMBER(10, 2);
3. rowcac NUMBER(10, 2);
4. bufcac NUMBER(10, 2);
5. redlog NUMBER(10, 2);
6. spsize NUMBER;
7. blkbuf NUMBER;
8. logbuf NUMBER;
9. BEGIN
10. SELECT VALUE
11. INTO redlog
12. FROM v$sysstat
13. WHERE name = 'redo log space requests';
14. SELECT 100 \* (SUM(pins) - SUM(reloads)) / SUM(pins)
15. INTO libcac
16. FROM v$librarycache;
18. SELECT 100 \* (SUM(gets) - SUM(getmisses)) / SUM(gets)
19. INTO rowcac
20. FROM v$rowcache;
21. SELECT 100 \* (cur.VALUE + con.VALUE - phys.VALUE) /(cur.VALUE + con.VALUE)
22. INTO bufcac
23. FROM v$sysstat cur, v$sysstat con, v$sysstat phys,
24. v$statname ncu, v$statname nco, v$statname nph
25. WHERE cur.statistic# = ncu.statistic#
26. AND ncu.name = 'db block gets'
27. AND con.statistic# = nco.statistic#
28. AND nco.name = 'consistent gets'
29. AND phys.statistic# = nph.statistic#
30. AND nph.name = 'physical reads';
31. SELECT VALUE INTO spsize
32. FROM v$parameter
33. WHERE name = 'shared\_pool\_size';
34. SELECT VALUE INTO blkbuf
35. FROM v$parameter
36. WHERE name = 'db\_block\_buffers';
37. SELECT VALUE INTO logbuf FROM v$parameter WHERE name = 'log\_buffer';
38. DBMS\_OUTPUT.put\_line('> SGA CACHE STATISTICS');
39. DBMS\_OUTPUT.put\_line('> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');
40. DBMS\_OUTPUT.put\_line('> SQL Cache Hit rate = ' || libcac);
41. DBMS\_OUTPUT.put\_line('> Dict Cache Hit rate = ' || rowcac);
42. DBMS\_OUTPUT.put\_line('> Buffer Cache Hit rate = ' || bufcac);
43. DBMS\_OUTPUT.put\_line('> Redo Log space requests = ' || redlog);
44. DBMS\_OUTPUT.put\_line('> ');
45. DBMS\_OUTPUT.put\_line('> INIT.ORA SETTING');
46. DBMS\_OUTPUT.put\_line('> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');
47. DBMS\_OUTPUT.put\_line('> Shared Pool Size = ' || spsize || ' Bytes');
48. DBMS\_OUTPUT.put\_line('> DB Block Buffer = ' || blkbuf || ' Blocks');
49. DBMS\_OUTPUT.put\_line('> Log Buffer = ' || logbuf || ' Bytes');
50. DBMS\_OUTPUT.put\_line('> ');
51. IF libcac < 99 THEN
52. DBMS\_OUTPUT.put\_line('\*\*\* HINT: Library Cache muito baixo! Aumente Shared Pool Size.');
53. END IF;
54. IF rowcac < 85 THEN
55. DBMS\_OUTPUT.put\_line('\*\*\* HINT: Row Cache muito baixo! Aumente Shared Pool Size.');
56. END IF;
57. IF bufcac < 90 THEN
58. DBMS\_OUTPUT.put\_line('\*\*\* HINT: Buffer Cache muito baixo! Aumente DB Block Buffer value.');
59. END IF;
60. IF redlog > 100 THEN
61. DBMS\_OUTPUT.put\_line('\*\*\* HINT: Valor de Log Buffer é muito baixo!');
62. END IF;
64. END;

## 6 – Views Materializadas

## Utilizamos elas para fazermos cálculos, armazenamentos de dados e dar agilidade na troca de informações entre um banco de dados ou entre tabelas. Este recurso é muito utilizado em ambientes de Data Warehouse, que trabalha com uma enorme quantidade de informações.

## A View Materializada deverá utilizar os dados imediatamente na query rewrite (Seu SELECT), desde modo os dados serão processados com mais agilidade.

## É necessário ter uma VIEW LOG nela.

## 7 – Indíces

2 tipos:

Primary Key , Unique Key

Manuallu:

Create Index, Alter table index

Índice baseado em Função -> Funções de grupo quebram o índice

CREATE INDEX index\_name on column\_table\_name(UPPER(Columns\_name))

## 8 – FlahsBack

Apagar a tabela TABLE\_NAME

DROP\_TABLE table\_name ;

DROP\_TABLE table\_name purge; -> Não vai para a lixeira

Recuperar os dados

Select \* from recyclebin; - Verificar Lixeira

FLASHBACK TABLE table\_name TO BEFORE DROP;

#Recuperação efetuado

#Limpar a lixeira

Purge recyclerbin

#Voltar o estado da tabela de 1 minuto atrás

FLASHBACK TABLE table\_name TO TIMESTAMOP (SYSTIMESTAMO – INTERVAL ‘1’ MINUTE)

#FlashBack de um Select

Select column\_name from table\_name VERSION BETWEEN SCN MINVALUE AND MAXVALUE where employee\_id = 107;

## 9 – Table Temporária

Create GLOBAL TEMPORARY TABLE table\_name on COMMIT DELETE ROWNS as select \* from table\_name

## 10 – Buscar todos os dicionários do Oracle

Select \* from dictionary

## 11 – Valores DEFAULT

Update table\_name set columns\_name = default

## 12 – Multiples Inserts

### Inser ALL

Insert ALL

INTO sal\_history values (campo1,campor2,campo3)

INTO mgr\_history values (campo1,campor2,campo3)

Select campo1,campo2,campor3 from table\_name where id>10

Insert all

When 1=1 then

Into t1 (campo 1, campo2) values (campo1,campo2)

When id<10 then

Into t2 (campo1, campo2) values (campo1,campor2)

Select campo1 , campo2 from table\_name

### Inser First ->

Mesma coisa do Insert ALL, porém, caso a primeira condição for verdadeira, não faz mais nada, outros Inserts, igual ao Insert ALL

Insert first

When id<10 then

Into t1 (campo 1, campo2) values (campo1,campo2)

When 1=1 then

Into t2 (campo1, campo2) values (campo1,campor2)

Select campo1 , campo2 from table\_name

### Insert ALL com Pivot

Transformar colunas em Linhas

INSERT ALL

Into sales\_info values(emplyee\_id, week\_id, sales\_mon)

Into sales\_info values(emplyee\_id, week\_id, sales\_TUE)

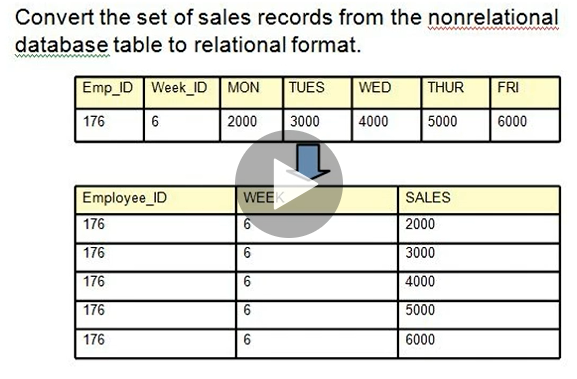
Into sales\_info values(emplyee\_id, week\_id, sales\_WED)

Into sales\_info values(emplyee\_id, week\_id, sales\_THUR)

Into sales\_info values(emplyee\_id, week\_id, sales\_FRI)

Select emplyee\_id, week\_id, sales\_mon, sales\_TUE, sales\_WED, sales\_THUR, sales\_FRI

From table\_name



## 13 – With

With t1 as (select campo1, campo2 from table\_name),

With t2 as (select campo3, campo4 from table\_name)

Select \* from t1 where t1.columns\_name = t2.columns\_name

## 14 – Expressões Regulares

## 15 – SYS\_REFCURSOR

CREATE OR REPLACE PROCEDURE PRC\_CURSOR\_EMPLOYEES

(pemployees\_cursor OUT SYS\_REFCURSOR)

IS

BEGIN

OPEN pemployees\_cursor

FOR

SELECT first\_name, last\_name

FROM employees;

END PRC\_CURSOR\_EMPLOYEES;

-- Procedure referenciando o Parametro OUT SYS\_REFCURSOR

CREATE OR REPLACE PROCEDURE PRC\_DISPLAY\_EMPOYEES

IS

employees\_cursor SYS\_REFCURSOR;

vfirst\_name employees.first\_name%TYPE;

vlast\_name employees.last\_name%TYPE;

BEGIN

PRC\_CURSOR\_EMPLOYEES(employees\_cursor);

LOOP

FETCH employees\_cursor

INTO vfirst\_name, vlast\_name;

EXIT WHEN employees\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(vfirst\_name || ' ' || vlast\_name);

END LOOP;

CLOSE employees\_cursor;

END PRC\_DISPLAY\_EMPOYEES;

-- Executando a Procedure PRC\_DISPLAY\_EMPOYEES

SET SERVEROUTPUT ON

SET VERIFY OFF

execute PRC\_DISPLAY\_EMPOYEES

## 16 – Builk Collect – For All e Limit

Builk Collect -> Realizar o Fetch das linhas recuperadas pelo comando SELECT e armazenar em uma Collection.

Serve para diminuir a troca de contextos.

Não é aconselhado para quando for muitas linhas. Se for muitas linhas, usar o Limit.

SET SERVEROUTPUT ON

SET VERIFY OFF

CREATE OR REPLACE PROCEDURE PRC\_UPDATE\_SALARY

(ppercentual IN NUMBER)

AS

TYPE employee\_id\_table\_type IS TABLE OF employees.employee\_id%TYPE

INDEX BY BINARY\_INTEGER; -- Type Associative Array

employee\_id\_table employee\_id\_table\_type;

BEGIN

SELECT DISTINCT employee\_id

BULK COLLECT INTO employee\_id\_table

FROM employees;

DBMS\_OUTPUT.PUT\_LINE('Linhas recuperadas: ' || employee\_id\_table.COUNT);

FOR /\*FORALL\*/ indice IN 1..employee\_id\_table.COUNT LOOP /\*ForAll é usado para diminuir a troca de contextos\*/

UPDATE employees e

SET e.salary = e.salary \* (1 + ppercentual / 100)

WHERE e.employee\_id = employee\_id\_table(indice); -- para cada UPDATE dentro do FOR LOOP Ocorrerá troca de contexto (Context Switch)

--

-- outro comandos

--

END LOOP; --Só tem o END LOOP se for o FOR . FORALL não usa o Loop;

END;

exec PRC\_UPDATE\_SALARY(10)

* Utilizando o Forall ocupa muito espaço de memória, buffer , por isso é bom usar o Limit.

CREATE OR REPLACE PROCEDURE PRC\_UPDATE\_SALARY2

(ppercentual IN NUMBER)

AS

**vLimit CONSTANT INTEGER(2) := 30;**

TYPE employee\_id\_table\_type IS TABLE OF employees.employee\_id%TYPE

INDEX BY BINARY\_INTEGER; -- Type Associative Array

employee\_id\_table employee\_id\_table\_type;

CURSOR employees\_cursor IS

SELECT employee\_id

FROM employees;

BEGIN

OPEN employees\_cursor;

LOOP

FETCH employees\_cursor

BULK COLLECT INTO employee\_id\_table **LIMIT vlimit;**

EXIT WHEN employee\_id\_table.COUNT = 0;

DBMS\_OUTPUT.PUT\_LINE('Linhas recuperadas: ' || employee\_id\_table.COUNT);

FORALL indice IN 1..employee\_id\_table.COUNT

UPDATE employees e

SET e.salary = e.salary \* (1 + ppercentual / 100)

WHERE e.employee\_id = employee\_id\_table(indice);

END LOOP;

CLOSE employees\_cursor;

-- COMMIT;

END;

17 – DBMS\_SCHEDULER

Serve para criação dos JOBS.