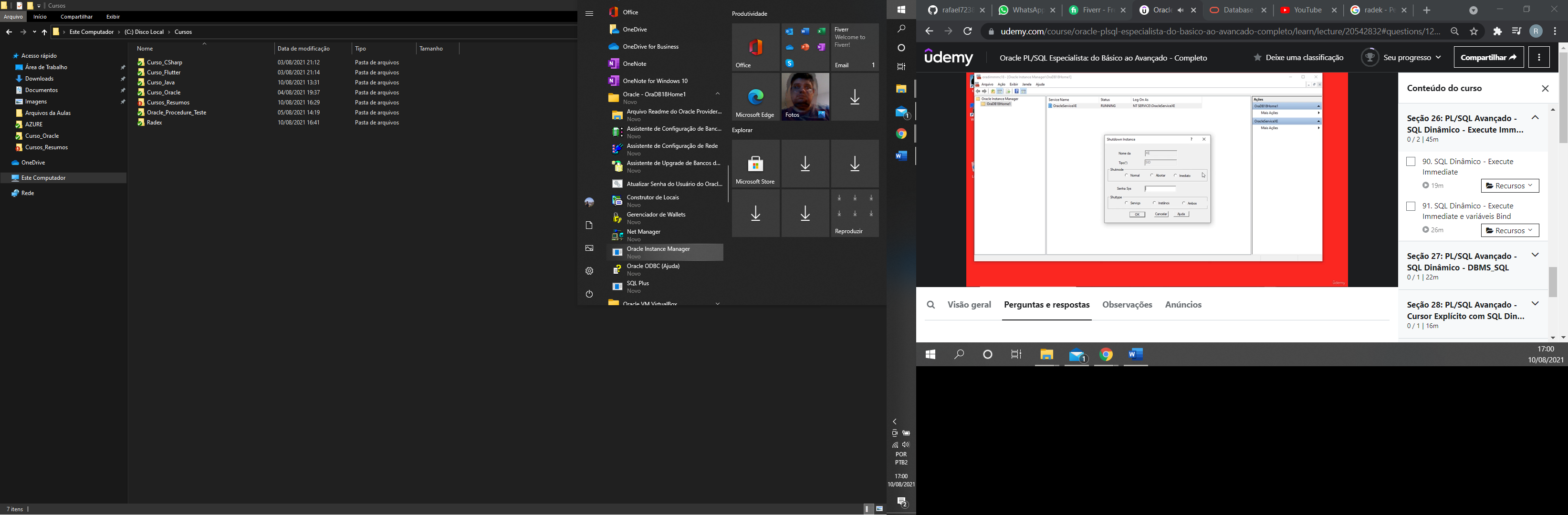
1 – Instalação – 18c XE.

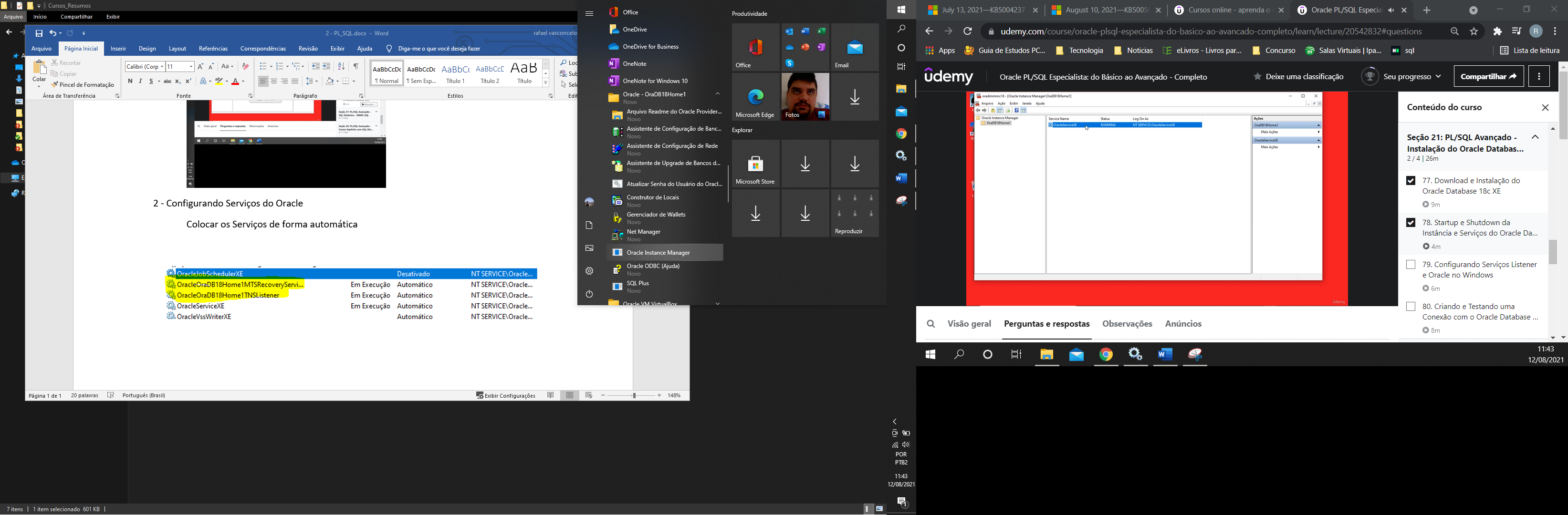
Instalação do banco.

<https://www.oracle.com/database/technologies/xe-downloads.html>

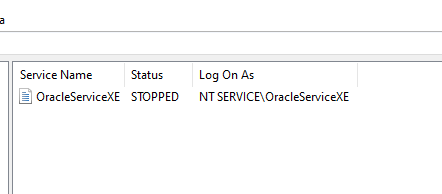


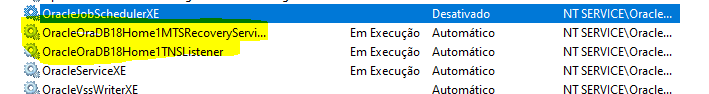
2 - Configurando Serviços do Oracle

Colocar os Serviços de forma automática



Aqui que a gente stopa o serviço do Oracle





Para agendamento de Jobs, precisa ativar esse serviço



Instalação do SQL DEVELOPER

<https://www.oracle.com/tools/downloads/sqldev-downloads.html>

# 3 – Visão Geral do PL/SQL

SET SERVEROUTPUT ON

DECLARE

vtexto VARCHAR2(32767) := 'Seja bem vindo ao PL/SQL';

BEGIN

dbms\_output.put\_line(vtexto);

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Erro Oracle: '

|| sqlcode

|| sqlerrm);

END;

# 4 – Bloco Anônimo

Começa com o DECLARE, BEGIN, EXCEPTION, END

SET SERVEROUTPUT ON

DECLARE

vtexto VARCHAR2(32767) := 'Seja bem vindo ao PL/SQL';

vnumero1 NUMBER(11, 2) := 500;

vnumero2 NUMBER(11, 2) := 500;

vresultado NUMBER(11, 2);

BEGIN

vresultado := ( vnumero1 + vnumero2 ) / 2;

dbms\_output.put\_line(vresultado);

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Erro Oracle: '

|| sqlcode

|| sqlerrm);

END;

# 5- Utilizando Variáveis

## Constantes

SET SERVEROUTPUT ON

DECLARE

vnumero1 CONSTANT NUMBER(11, 2) := 1200.50;

BEGIN

--vnumero1:=0;

dbms\_output.put\_line(vnumero1);

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Erro Oracle: '

|| sqlcode

|| sqlerrm);

END;

## DataTypes

Long Raw -> Armazenar arquivo, foto. São arquivos binários.

RowId -> Caracter de 18 campos.

Timestamp -> Armazena uma data até 9 dígitos de segundos após o decimal.

DECLARE

vData1 TIMESTAMP := SYSTIMESTAMP;

Timestamp com Time Zone -> Armazena a data UTC

DECLARE

Vdata1 TIMESTAMP WITH TIME ZONE:= SYSTIMESTAMP;

Timestamo com Local Time Zone -> Time do Client

Vdata2 TIMESTAMO WITH LOCAL TIME ZONE:= SYSTIMESTAMP;

BINARY\_INTEGER => Armazenamento de Dados inteiros, sem casas decimais.

Vvar1 BINARY\_INTEGER:=4;

BINARY\_FLOAT -> são opções do Number. 32 bits

BINARY\_DOUBLE -> são opções do Number. 64 bits.

Variáveis por Referência -> %Type

vfistName employees.first\_name%type;

Bind -> Variáveis declaradas em ambiente externo ao bloco.

SET SERVEROUTPUT ON

VARIABLE gmedia NUMBER;

DECLARE

vnumero1 NUMBER(11,2):=2000;

vnumero2 NUMBER(11,2):=4000;

BEGIN

:gmedia:=(vnumero1+vnumero2)/2;

dbms\_output.put\_line('valor é: '|| :gmedia);

END;

Declarar Variável por referência (%Type)

Declare

vFirst\_name employess.first\_name%TYPE;

Prefixando uma variável com a label do bloco:

<<Bloco1>>

Declare

Vbloco1 varchar(30) :=’bloco 1’;

Begin

Dbms.output\_putline(bloco1.vbloco1);

<<bloco2>>

Declare

Vbloco1 varchar(30) :=’bloco 1’;

Begin

Dbms.output\_putline(bloco1.vbloco1);

Dbms.output\_putline(bloco2.vbloco1);

End;

End;

# 6 – Utilizando Select, Insert, Update, Delete, SavePoint no PL/SQL

No Select deve retornar apenas umas linha;

#Aqui vai inserir um registro. Salva o ponto de restauração. Faz um Update. Porém, tudo que foi feito entre o ponto de restauração até o momento, será dado RollBack. O Commit apenas considerará o Insert

BEGIN

INSERT INTO EMPLOYEES (XXXXX) VALUES (XXXXX);

SAVEPOINT INSERTOK;

UPDATE employees set salary=5000 where job\_id='IT\_PROG' ;

ROLLBACK TO INSERTOK;

COMMIT;

END;

# 7 – Cursor Implícito

Sempre que executa um comando SQL, o servidor Oracle abre uma memória na qual o comando é analisado. Chamado de Implícito.

4 Atributos de Cursor:

Tabela

Descrição gerada automaticamente

O SQL%ISOPEN sempre retornará falso.

SET SERVEROUTPUT ON

DECLARE

vDepartamento\_id employees.department\_id%type :=40;

vPercentual NUMBER(3):=10;

BEGIN

UPDATE employees set salary=salary \* (1\*vpercentual/100) where department\_id=vdepartamento\_id;

dbms\_output.put\_line('Número de Atualizações de emrpegados atualizado: '|| SQL%ROWCOUNT);

-- dbms\_output.put\_line('Linhas Afetadas TRUE or FALSE ' || SQL%NOTFOUND);

IF SQL%FOUND THEN

dbms\_output.put\_line('Linhas Afetadas TRUE or FALSE ' );

END IF;

ROLLBACK;

-- COMMIT;

END;

# 8 IF

SET SERVEROUTPUT ON

ACCEPT pdepartment\_id PROMPT 'Digite o ID do Departamento: '

DECLARE

vpercentual NUMBER(3);

vdepartament\_id employees.department\_id%TYPE := &pdepartment\_id;

BEGIN

IF vdepartament\_id = 80 THEN

vpercentual:=10;

-- vpercentual:=15;

ELSE

vpercentual := 20;

END IF;

DBMS\_OUTPUT.put\_line('Departament ID: '||vdepartament\_id);

DBMS\_OUTPUT.put\_line('Percentual : '||vpercentual);

END;

# 9 Case

## Exemplo 1:

SET SERVEROUTPUT ON

ACCEPT pdepartment\_id PROMPT 'Digite o ID do Departamento: ';

DECLARE

vpercentual NUMBER(3);

vdepartament\_id employees.department\_id%TYPE := &pdepartment\_id;

BEGIN

CASE vdepartament\_id

WHEN 80 THEN

vpercentual := 10;

WHEN 60 THEN

vpercentual := 15;

ELSE

vpercentual := 20;

END CASE;

dbms\_output.put\_line('Departament ID: ' || vdepartament\_id);

dbms\_output.put\_line('Percentual : ' || vpercentual);

END;

## Exemplo 2:

SET SERVEROUTPUT ON

ACCEPT pdepartment\_id PROMPT 'Digite o ID do Departamento: ';

DECLARE

vpercentual NUMBER(3);

vdepartament\_id employees.department\_id%TYPE := &pdepartment\_id;

BEGIN

CASE

WHEN vdepartament\_id=80 THEN

vpercentual := 10;

WHEN vdepartament\_id=60 THEN

vpercentual := 15;

ELSE

vpercentual := 20;

END CASE;

dbms\_output.put\_line('Departament ID: ' || vdepartament\_id);

dbms\_output.put\_line('Percentual : ' || vpercentual);

END;

# 10 – Loop básico

SET SERVEROUTPUT ON

ACCEPT plimite PROMPT 'Digite o Valor do Limite: ';

DECLARE

vnumero NUMBER(3) := 1;

vlimite NUMBER := &plimite;

BEGIN

LOOP

dbms\_output.put\_line('Número : ' || to\_char(vnumero));

EXIT WHEN vnumero = vlimite;

vnumero := vnumero + 1;

END LOOP;

END;

# 11 – FOR LOOP

SET SERVEROUTPUT ON

ACCEPT plimite PROMPT 'Digite o Valor do Limite: ';

DECLARE

vinicio NUMBER(3) := 1;

vfim NUMBER := &plimite;

BEGIN

FOR i IN vinicio..vfim LOOP

dbms\_output.put\_line('Número = ' || i);

END LOOP;

END;

# 12 – While Loop

SET SERVEROUTPUT ON

ACCEPT plimite PROMPT 'Digite o Valor do Limite: ';

DECLARE

vNumero NUMBER(3) := 1;

vLimite NUMBER := &plimite;

BEGIN

WHILE vNumero<=vLimite LOOP

dbms\_output.put\_line('Número = ' || vNumero);

vNumero:=vNumero+1;

END LOOP;

END;

# 13 – Saindo do Loop

Posso usar o Label para sair do Loop principal;

SET SERVEROUTPUT ON

DECLARE

vsoma NUMBER := 1;

BEGIN

<<loop1>> FOR i IN 1..8 LOOP

<<loop2>> FOR y IN 1..8 LOOP

dbms\_output.put\_line(vsoma);

vsoma := vsoma \* 3;

-- EXIT WHEN vsoma>68630377364883;

EXIT loop1 WHEN vsoma>68630377364883;

END LOOP;

END LOOP;

END;

# 14 – Variável PL/SQL Records - %type

Uma variável que pode se dividir em campos

SET SERVEROUTPUT ON

SET VERIFY OFF

ACCEPT pemployee\_id PROMPT 'Digite o ID do empregado: '

DECLARE

TYPE employee\_record\_type IS RECORD (

employee\_id employees.employee\_id%TYPE,

first\_name employees.first\_name%TYPE

);

employee\_record employee\_record\_type;

BEGIN

SELECT employee\_id, first\_name

INTO employee\_record

FROM employees

WHERE employee\_id = &pemployee\_id;

DBMS\_OUTPUT.put\_line(employee\_record.employee\_id || ' '||employee\_record.first\_name );

END;

# 15 – Record com %RowType

Usado para um campo ter as mesmas características de uma tabela:

SET SERVEROUTPUT ON

SET VERIFY OFF

ACCEPT pemployee\_id PROMPT 'Digite o ID do empregado: '

DECLARE

employee\_record employees%rowtype;

BEGIN

SELECT

\*

INTO

employee\_record

FROM

employees

WHERE

employee\_id = &pemployee\_id;

DBMS\_OUTPUT.put\_line(employee\_record.employee\_id || ' '||employee\_record.first\_name );

END;

# 16 – Collection – Associative Arrays

Gerenciar múltiplas linhas de dados

São Listas(Vetores).

Associative Arrays -> Obrigatório o uso do Index-by-tables.

Pode ser indexado por number ou arrays.

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

TYPE numero\_table\_type IS

TABLE OF NUMBER(2) INDEX BY BINARY\_INTEGER;

numero\_table numero\_table\_type;

BEGIN

FOR i IN 1..10 LOOP

numero\_table(i) := i;

END LOOP;

FOR i IN 1..numero\_table.COUNT() LOOP

DBMS\_OUTPUT.PUT\_LINE(numero\_table(i));

END LOOP;

END;

# 17 – Collection – Associative Arrays de Records

Usar o Associative Array na tabela

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

TYPE employees\_table\_type IS

TABLE OF employees%rowtype

INDEX BY BINARY\_INTEGER;

employees\_table employees\_table\_type;

BEGIN

SELECT \* BULK COLLECT INTO employees\_table from employees;

--Cuidado ao Usar o BULK COLLECT....Sempre usar com o FORALL e o LIMIT

FOR i IN employees\_table.FIRST..employees\_table.LAST LOOP

DBMS\_OUTPUT.PUT\_LINE(employees\_table(i).first\_name);

END LOOP;

END;

# 18 – Collection – Nested Table

Para alocar usa o EXTEND

O índice tem que ser positivo

Não tem o Index By

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

TYPE numero\_table\_type IS

TABLE OF INTEGER(2);

numero\_table numero\_table\_type := numero\_table\_type();

BEGIN

FOR i IN 1..10 LOOP

numero\_table.extend;

numero\_table(i) := i;

dbms\_output.put\_line(i);

END LOOP;

FOR i IN 1..numero\_table.COUNT LOOP

dbms\_output.put\_line(numero\_table(i));

END LOOP;

END;

# 19 – Collection – Nested Table of Record

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

TYPE employees\_table\_type IS

TABLE OF employees%rowtype;

employees\_table employees\_table\_type := employees\_table\_type(); --Sem o Index By

BEGIN

SELECT \* BULK COLLECT INTO employees\_table from employees;

FOR i IN employees\_table.FIRST.. employees\_table.LAST LOOP

dbms\_output.put\_line(employees\_table(i).first\_name);

END LOOP;

END;

# 20 – Collection – VARRAY

Precisa definir um tamanho máximo.

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

TYPE Numero\_table\_type IS

VARRAY (10) of INTEGER (2);

Numero\_table Numero\_table\_type := Numero\_table\_type();

BEGIN

FOR i IN 1..10 LOOP

Numero\_table.EXTEND;

Numero\_table(i):=i;

END LOOP;

FOR i IN 1..10 LOOP

DBMS\_OUTPUT.PUT\_LINE( Numero\_table(i));

END LOOP;

END;

# 21 – Collection – VARRAY of Records

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

TYPE employees\_table\_type IS

VARRAY(200) OF employees%rowtype;

employees\_table employees\_table\_type := employees\_table\_type();

BEGIN

SELECT \* BULK COLLECT INTO employees\_table from employees;

FOR i IN employees\_table.FIRST.. employees\_table.LAST LOOP

dbms\_output.put\_line(employees\_table(i).first\_name);

END LOOP;

END;.

Tabela

Descrição gerada automaticamente

Tabela

Descrição gerada automaticamente

Tabela

Descrição gerada automaticamente

# 22 – Cursor Explícito

Private SQL Areas -> armazena informações do processamento

Passos: DECLARE -> OPEN -> FETCH -> CLOSE

Tabela

Descrição gerada automaticamente

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

CURSOR employees\_cursor IS

SELECT \* FROM employees;

employees\_record employees\_cursor%rowtype; --Mesma estrutura do Cursor

BEGIN

OPEN employees\_cursor;

LOOP

FETCH employees\_cursor INTO employees\_record;

EXIT WHEN employees\_cursor%notfound;

DBMS\_OUTPUT.PUT\_LINE(employees\_record.first\_name);

END LOOP ;

CLOSE employees\_cursor;

END;

# 23 – Cursor Com Parâmetros

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

CURSOR employees\_cursor (pdepartament\_id NUMBER) IS

SELECT \* FROM employees where DEPARTMENT\_ID=pdepartament\_id;

employees\_record employees\_cursor%rowtype; --Mesma estrutura do Cursor

BEGIN

OPEN employees\_cursor(126);

LOOP

FETCH employees\_cursor INTO employees\_record;

IF employees\_cursor%notfound THEN

DBMS\_OUTPUT.PUT\_LINE('Não veio Registro');

END IF;

EXIT WHEN employees\_cursor%notfound;

DBMS\_OUTPUT.PUT\_LINE(employees\_record.first\_name);

END LOOP ;

CLOSE employees\_cursor;

END;

# 24 – CURSOR for update

For update -> Faz block nas linhas encontradas no Select

SET SERVEROUT ON

SET VERIFY OFF

DECLARE

CURSOR employees\_cursor (pdepartament\_id NUMBER) IS

SELECT \* FROM employees where DEPARTMENT\_ID=pdepartament\_id for update;

employees\_record employees\_cursor%rowtype; --Mesma estrutura do Cursor

BEGIN

OPEN employees\_cursor(126);

LOOP

FETCH employees\_cursor INTO employees\_record;

IF employees\_cursor%notfound THEN

DBMS\_OUTPUT.PUT\_LINE('Não veio Registro');

END IF;

EXIT WHEN employees\_cursor%notfound;

DBMS\_OUTPUT.PUT\_LINE(employees\_record.first\_name);

END LOOP ;

CLOSE employees\_cursor;

END;

# 25 – Exceções

DECLARE -> BEGIN -> EXCEPTION -> END;

Interromper a execução de um programa PL/SQL através de uma chamada a procedure pré-definida ORACLE RAISE\_APPLICATION\_ERROR.

SQLCODE -> Código de erro do ORACLE.

SQLERRM -> Mensagem de erro.

--ACCEPT pEmplyee\_id PROMPT 'Digite o Id do Empregado: ';

SET VERIFY OFF

DECLARE

vFirst\_name employees.first\_name%type;

BEGIN

select first\_name into vFirst\_name from employees where rownum<=0;

EXCEPTION

WHEN OTHERS THEN

rollback;

RAISE\_APPLICATION\_ERROR(-20000, 'Código Erro: '||SQLCODE ||' Mensagem Erro: '||SQLERRM);

END;

# 26 – Criar Minhas Exceções

SET VERIFY OFF;

DECLARE

vfirst\_name employees.first\_name%TYPE;

exminhaexcessao EXCEPTION;

BEGIN

SELECT

first\_name

INTO vfirst\_name

FROM

employees

WHERE

ROWNUM <= 1;

IF vfirst\_name = 'Ellen' THEN

RAISE exminhaexcessao;

END IF;

EXCEPTION

WHEN exminhaexcessao THEN

raise\_application\_error(-20000, 'Foi para a minha Excessão......Código Erro: '

|| sqlcode

|| ' Mensagem Erro: '

|| sqlerrm);

WHEN OTHERS THEN

raise\_application\_error(-20000, 'Código Erro: '

|| sqlcode

|| ' Mensagem Erro: '

|| sqlerrm);

END;

# 27 – Procedures

CREATE OR REPLACE PROCEDURE prc\_insere\_empregado (

pfirstname IN VARCHAR2,

plastname OUT VARCHAR2

) IS

--Coloco as variáveis aqui

BEGIN

dbms\_output.put\_line(pfirstname);

plastname := 'Vasconcelos';

EXCEPTION

WHEN OTHERS THEN

BEGIN

raise\_application\_error(-2002, 'ERRO');

END;

END;

SET SERVEROUT ON

DECLARE

employees\_record employees%rowtype;

BEGIN

prc\_insere\_empregado('TESTE2', employees\_record.last\_name);

dbms\_output.put\_line(employees\_record.last\_name);

END;

# 28 - Funções

create or replace FUNCTION fnc\_consulta\_salario\_empregado (

pEmployees\_id IN NUMBER

) RETURN NUMBER IS

vsalary employees.salary%TYPE;

BEGIN

select salary into vsalary from employees where employee\_id = pEmployees\_id;

return vsalary;

EXCEPTION

WHEN OTHERS THEN

BEGIN

raise\_application\_error(-2002, 'ERRO');

END;

END;

SET SERVEROUT ON

DECLARE

v\_return NUMBER;

BEGIN

v\_return := fnc\_consulta\_salario\_empregado(128);

dbms\_output.put\_line('v\_Return = ' || v\_return);

END;

# 29 – Mostrar os Erros dos Objetos

Select \* from dba\_Erros

# 30 – Dependências

select \* from all\_dependencies

# 31 – Deptree / Ideptree

São visões que servem para mostrar a hierárquia das dependências

Executar o comando para criar a Procedure de Hierarquia: @C:\app\rafae\product\18.0.0\dbhomeXE\rdbms\admin\utldtree.sql

/\*Verifica onde a tabela EMPLOYEES usa\*/

EXEC deptree\_fill('TABLE','HR','EMPLOYEES');

select \* from deptree order by nested\_level;

# 32 – Debug Procedure ou Function

Componentes **públicos** de uma Package

Basta declarar na Package Specificatio

Variável Global

Componentes **privados** de uma Package

Só pode referenciar dentreo da Package.

Basta não definir na Package Specification

Texto

Descrição gerada automaticamente

create or replace PACKAGE PCK\_EMPREGADOS

IS

gMinSalary employees.salary%TYPE;

PROCEDURE PRC\_INSERE\_EMPREGADO

(pfirst\_name IN VARCHAR2,);

PROCEDURE PRC\_AUMENTA\_SALARIO\_EMPREGADO

(pemployee\_id IN NUMBER,

ppercentual IN NUMBER);

FUNCTION FNC\_CONSULTA\_SALARIO\_EMPREGADO

(pemployee\_id IN NUMBER)

RETURN NUMBER;

END PCK\_EMPREGADOS;

create or replace PACKAGE BODY PCK\_EMPREGADOS

IS

PROCEDURE PRC\_INSERE\_EMPREGADO

(pfirst\_name IN VARCHAR2)

IS

BEGIN

IF PCK\_EMPREGADOS.gMinSalary<2000 then

RAISE\_APPLICATION\_ERROR(-2002,'Salário não pode ser menor que o máximo');

END IF;

INSERT INTO employees (

first\_name)

VALUES (

pfirst\_name );

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Erro Oracle ' || SQLCODE || SQLERRM);

END;

PROCEDURE PRC\_AUMENTA\_SALARIO\_EMPREGADO

(pemployee\_id IN NUMBER)

IS

-- Nenhuma váriável declarada

BEGIN

UPDATE employees

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

WHERE employee\_id = pemployee\_id;

EXCEPTION

WHEN OTHERS

THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Erro Oracle: ' || SQLCODE || ' - ' || SQLERRM);

END;

FUNCTION FNC\_CONSULTA\_SALARIO\_EMPREGADO

(pemployee\_id IN NUMBER)

RETURN NUMBER

IS

vsalary employees.salary%TYPE;

BEGIN

SELECT salary

INTO vsalary

FROM employees

WHERE employee\_id = pemployee\_id;

RETURN (vsalary);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Empregado inexistente');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Erro Oracle ' || SQLCODE || SQLERRM);

END;

BEGIN

Select max(salary) into PCK\_EMPREGADOS.gMinSalary; --Toda vez que executar alguma function ou Procedure

--dessa Package, o que está entre o BEGIN e END será executado

-- E a informação será executada; Variável gravada no specific

END PCK\_EMPREGADOS;

# 33 – Triggers

Triggers para Tabelas e Views

Triggers por padrão não fazem controle de transação (Commit, Roolback e SavePoint), ao menos que seja declarado o PRAGMA AUTONOMOUS\_TRANSACTION.

Elementos de uma DML TRIGGER

Tabela

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

## Regras do Mutante Table:

Não pode , na Trigger, fazer comandos DML na tabela que está sendo mexida, para os campos, Chave Primária, Estrangeira, Chaves únicas (Unique).

Não se pode fazer SELECT na tabela mutande;

Solução:

Chamar uma Procedure/Package que faça isso; OU

:new.cheack:=’fasdfasdfasd’;

-- Resolvendo o problema de Mutating Tables

CREATE OR REPLACE PACKAGE PCK\_EMPLOYEES\_DADOS

AS

TYPE max\_salary\_table\_type IS TABLE OF NUMBER(10,2)

INDEX BY BINARY\_INTEGER;

gMaxSalary max\_salary\_table\_type;

END PCK\_EMPLOYEES\_DADOS;

CREATE OR REPLACE TRIGGER B\_IU\_VALIDATE\_SALARY\_EMPLOYEES\_S\_TRG

BEFORE INSERT OR UPDATE OF salary

ON employees

DECLARE

vMaxSalary employees.salary%TYPE;

BEGIN

SELECT MAX(salary)

INTO PCK\_EMPLOYEES\_DADOS.gMaxSalary(1)

FROM employees;

END;

CREATE OR REPLACE TRIGGER B\_IU\_VALIDATE\_SALARY\_EMPLOYEES\_R\_TRG

BEFORE INSERT OR UPDATE OF salary

ON employees

FOR EACH ROW

DECLARE

vMaxSalary employees.salary%TYPE;

BEGIN

IF :new.salary > PCK\_EMPLOYEES\_DADOS.gMaxSalary(1) \* 1.2

THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Novo salario não pode ser superior ao maior salario + 20%');

END IF;

END;

-- Testando Violação da Regra 2

SET VERIFY OFF

UPDATE employees

SET salary = 30000

WHERE employee\_id = 100;

COMMIT;

# 34 – Sys\_REFCURSOR

Aponta para um cursor ou para um result set de um cursor

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;

CREATE OR REPLACE PROCEDURE prc\_display\_empoyees IS

employees\_cursor SYS\_REFCURSOR;

vfirst\_name employees.last\_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);

END LOOP;

CLOSE employees\_cursor;

END prc\_display\_empoyees;

SET SERVEROUT ON

EXEC prc\_display\_empoyees;

# 35 – Builk Collect e FOR ALL

Diminuir a troca de contexto. Porque o PL/SQL Engienner vai executar apenas um contexto de SQL Engineer.

FOR ALL -> Permite apenas um comando DML.

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);

FORALL indice IN 1..employee\_id\_table.COUNT -- FOR ALL empacota todos os UPDATES e envia o pacote em 1 única troca de contexto (Context Switch)

UPDATE employees e

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

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

END;

# 36 – BUILD COLLECT com 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;

# 37 – Execute Imediate

CREATE OR REPLACE PROCEDURE PRC\_FETCH\_EMPLOYEES\_DYNAMIC

(pmanager\_id IN employees.manager\_id%TYPE DEFAULT NULL,

pdepartment\_id IN employees.department\_id%TYPE DEFAULT NULL)

AS

vemployees\_record employees%ROWTYPE;

vsql VARCHAR2(600) := 'SELECT \*

FROM employees

WHERE 1=1 ';

TYPE employees\_table\_type IS TABLE OF employees%ROWTYPE -- Associative Array

INDEX BY PLS\_INTEGER;

employees\_table employees\_table\_type;

BEGIN

IF pmanager\_id IS NOT NULL THEN

vsql := vsql || ' AND manager\_id = ' || pmanager\_id;

END IF;

IF pdepartment\_id IS NOT NULL THEN

vsql := vsql || ' AND department\_id = ' || pdepartment\_id;

END IF;

DBMS\_OUTPUT.PUT\_LINE(vsql);

EXECUTE IMMEDIATE vsql

BULK COLLECT INTO employees\_table;

FOR i IN 1..employees\_table.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE(employees\_table(i).employee\_id);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001,'Erro Oracle ' || SQLCODE || SQLERRM);

END;

-- Executando a procedure

exec PRC\_FETCH\_EMPLOYEES\_DYNAMIC(pmanager\_id => 103, pdepartment\_id => 60)

exec PRC\_FETCH\_EMPLOYEES\_DYNAMIC(pmanager\_id => 103)

# 38 – Execute Imediate – Variáveis BIND

Mais rápido porque não vai para o Pool. Utiliza o mesmo plano de execução.

SET SERVEROUTPUT ON

SET VERIFY OFF

CREATE OR REPLACE PROCEDURE PRC\_FETCH\_EMPLOYEES\_DYNAMIC\_BIND

(pmanager\_id IN employees.manager\_id%TYPE DEFAULT NULL,

pdepartment\_id IN employees.department\_id%TYPE DEFAULT NULL)

AS

vemployees\_record employees%ROWTYPE;

vsql VARCHAR2(600) := 'SELECT \*

FROM employees

WHERE 1=1 ';

TYPE employees\_table\_type IS TABLE OF employees%ROWTYPE -- Associative Array

INDEX BY PLS\_INTEGER;

employees\_table employees\_table\_type;

BEGIN

IF pmanager\_id IS NOT NULL THEN

vsql := vsql || ' AND manager\_id = :manager\_id';

END IF;

IF pdepartment\_id IS NOT NULL THEN

vsql := vsql || ' AND department\_id = :department\_id';

END IF;

DBMS\_OUTPUT.PUT\_LINE(vsql);

CASE

WHEN pmanager\_id IS NOT NULL AND pdepartment\_id IS NOT NULL THEN

EXECUTE IMMEDIATE vsql BULK COLLECT INTO employees\_table USING pmanager\_id, pdepartment\_id;

WHEN pmanager\_id IS NOT NULL AND pdepartment\_id is NULL THEN

EXECUTE IMMEDIATE vsql BULK COLLECT INTO employees\_table USING pmanager\_id;

WHEN pmanager\_id IS NULL AND pdepartment\_id IS NOT NULL THEN

EXECUTE IMMEDIATE vsql BULK COLLECT INTO employees\_table USING pdepartment\_id;

ELSE

EXECUTE IMMEDIATE vsql BULK COLLECT INTO employees\_table;

END CASE;

FOR i IN 1..employees\_table.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE(employees\_table(i).employee\_id || ' - ' ||

employees\_table(i).first\_name || ' - ' ||

employees\_table(i).last\_name || ' - ' ||

employees\_table(i).email || ' - ' ||

employees\_table(i).manager\_id || ' - ' ||

employees\_table(i).department\_id);

END LOOP;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001,'Erro Oracle ' || SQLCODE || SQLERRM);

END;

exec PRC\_FETCH\_EMPLOYEES\_DYNAMIC\_BIND(pmanager\_id => 103, pdepartment\_id => 60)

exec PRC\_FETCH\_EMPLOYEES\_DYNAMIC\_BIND(pmanager\_id => 103)

# 39 – Cursor Explícito – SQL Dinâmico

CREATE OR REPLACE PROCEDURE PRC\_FETCH\_EMPLOYEES\_CURSOR\_DYNAMIC

(pmanager\_id IN employees.manager\_id%TYPE DEFAULT NULL,

pdepartment\_id IN employees.department\_id%TYPE DEFAULT NULL)

AS

TYPE employees\_cursor\_type IS REF CURSOR;

employees\_cursor employees\_cursor\_type;

vemployees\_record employees%ROWTYPE;

vsql VARCHAR2(600) := 'SELECT \*

FROM employees

WHERE 1=1 ';

BEGIN

IF pmanager\_id IS NOT NULL THEN

vsql := vsql || ' AND manager\_id = :manager\_id';

END IF;

IF pdepartment\_id IS NOT NULL THEN

vsql := vsql || ' AND department\_id = :department\_id';

END IF;

DBMS\_OUTPUT.PUT\_LINE(vsql);

CASE

WHEN pmanager\_id IS NOT NULL AND pdepartment\_id IS NOT NULL THEN

OPEN employees\_cursor FOR vsql USING pmanager\_id, pdepartment\_id;

WHEN pmanager\_id IS NOT NULL AND pdepartment\_id is NULL THEN

OPEN employees\_cursor FOR vsql USING pmanager\_id;

WHEN pmanager\_id IS NULL AND pdepartment\_id IS NOT NULL THEN

OPEN employees\_cursor FOR vsql USING pdepartment\_id;

ELSE

OPEN employees\_cursor FOR vsql;

END CASE;

LOOP

FETCH employees\_cursor

INTO vemployees\_record;

EXIT WHEN employees\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(vemployees\_record.employee\_id);

END LOOP;

CLOSE employees\_cursor;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001,'Erro Oracle ' || SQLCODE || SQLERRM);

END;

exec PRC\_FETCH\_EMPLOYEES\_CURSOR\_DYNAMIC(pmanager\_id => 103, pdepartment\_id => 60)

exec PRC\_FETCH\_EMPLOYEES\_CURSOR\_DYNAMIC(pmanager\_id => 103)

# 40 – DBMS\_SCHEDULER

Criação de tarefefas(Jobs) Agendados.

Pode executar uma procedure, C, Java, Shell Linux.

Sintaxe:

Texto

Descrição gerada automaticamente

Program\_type

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

## Criando o programa

CREATE OR REPLACE PROCEDURE PRC\_INSERE\_DATA\_AGENDA

IS

BEGIN

INSERT INTO hr.agenda

VALUES (agenda\_seq.NEXTVAL, sysdate);

COMMIT;

END;

-- Criando e Habilitado um Programa

BEGIN

DBMS\_SCHEDULER.create\_program(

program\_name => 'HR.PRC\_INSERE\_AGENDA',

program\_action => 'HR.PRC\_INSERE\_DATA\_AGENDA', --Nome da Procedure

program\_type => 'STORED\_PROCEDURE',

number\_of\_arguments => 0,

comments => 'Insere dados na agenda',

enabled => TRUE);

/\*

DBMS\_SCHEDULER.ENABLE(name=>'HR.PRC\_INSERE\_AGENDA');

\*/

END;

## Criando uma Agenda (Scheduler)

Sintaxe:

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Porque o intervalo de repetição ficou em branco, considerando 1.

Texto

Descrição gerada automaticamente

grant CREATE JOB to hr;

BEGIN

DBMS\_SCHEDULER.CREATE\_SCHEDULE (

schedule\_name => 'SCH\_A\_CADA\_10\_SEGUNDOS',

start\_date => SYSTIMESTAMP,

--start\_date => TO\_TIMESTAMP\_TZ('2020-03-17 15:17:36.000000000 AMERICA/SAO\_PAULO','YYYY-MM-DD HH24:MI:SS.FF TZR'),

repeat\_interval => 'FREQ=SECONDLY;INTERVAL=10',

end\_date => TO\_TIMESTAMP\_TZ('2020-07-23 23:00:00.000000000 AMERICA/SAO\_PAULO','YYYY-MM-DD HH24:MI:SS.FF TZR'),

comments => 'A cada 10 segundos'

);

END;

## Criando o Job

Associa o Programa ao Scheduler

Sintaxe:

Texto

Descrição gerada automaticamente

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB (

job\_name => '"HR"."JOB\_INSERE\_DATA\_AGENDA"',

program\_name => '"HR"."PRC\_INSERE\_AGENDA"',

schedule\_name => '"HR"."SCH\_A\_CADA\_10\_SEGUNDOS"',

enabled => TRUE,

auto\_drop => FALSE,

comments => 'Job Insere Data na Agenda',

job\_style => 'REGULAR');

END;

BEGIN

DBMS\_SCHEDULER.DROP\_JOB (

job\_name => '"HR"."JOB\_INSERE\_DATA\_AGENDA"',

force => TRUE);

END;

# 41 - UTL\_FILE

Ler ou gravar um arquivo.

1. Criar um diretório
2. Fazer um grant nesse diretório

## --Escrevendo o Arquivo

CREATE OR REPLACE DIRECTORY ARQUIVOS AS 'C:\ARQUIVOS'; --DBA

GRANT READ, WRITE ON DIRECTORY ARQUIVOS TO hr; --DBA

SET SERVEROUTPUT ON

SET VERIFY OFF

DECLARE

vfile UTL\_FILE.FILE\_TYPE;

CURSOR employees\_c IS

SELECT employee\_id, first\_name, last\_name, job\_id, salary

FROM employees;

BEGIN

vfile := UTL\_FILE.FOPEN( ‘ARQUIVOS’, 'employees.txt','w',32767); --Função para abrir o Arquivo. Parâmetros => Nome do Diretório ;Nome do Arquivo ; w-> Escrita; Buffer

FOR employees\_r IN employees\_c LOOP

UTL\_FILE.PUT\_LINE(vfile, employees\_r.employee\_id || ';' ||

employees\_r.first\_name || ';' ||

employees\_r.last\_name || ';' ||

employees\_r.job\_id || ';' ||

employees\_r.salary);

END LOOP;

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Arquivo Texto employees.txt gerado com sucesso');

EXCEPTION

WHEN UTL\_FILE.INVALID\_PATH THEN

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Diretório Inválido');

WHEN UTL\_FILE.INVALID\_OPERATION THEN

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Operação invalida no arquivo');

WHEN UTL\_FILE.WRITE\_ERROR THEN

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Erro de gravação no arquivo');

WHEN UTL\_FILE.INVALID\_MODE THEN

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Modo de acesso inválido');

WHEN OTHERS THEN

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Erro Oracle:' || SQLCODE || SQLERRM);

END;

## -----Lendo o Arquivo

SET SERVEROUTPUT ON

SET VERIFY OFF

DECLARE

vfile UTL\_FILE.FILE\_TYPE;

vregistro VARCHAR2(400);

BEGIN

vfile := UTL\_FILE.FOPEN('ARQUIVOS', 'employees.txt','r',32767); --Nome do Diretório

LOOP

UTL\_FILE.GET\_LINE(vfile, vregistro);

DBMS\_OUTPUT.PUT\_LINE(vregistro);

END LOOP;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Arquivo Texto employees.txt lido com sucesso');

WHEN UTL\_FILE.INVALID\_PATH THEN

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Diretório Inválido');

WHEN OTHERS THEN

UTL\_FILE.FCLOSE(vfile);

DBMS\_OUTPUT.PUT\_LINE('Erro Oracle:' || SQLCODE || SQLERRM);

END;

# 42 – FlashBack

## Package DBMS\_FLASHBACK

grant execute on DBMS\_FLASHBACK to hr;

SELECT employee\_id, first\_name, last\_name, job\_id, salary

FROM employees

WHERE job\_id = 'IT\_PROG';

UPDATE employees

SET salary = salary \* 2

WHERE job\_id = 'IT\_PROG';

SELECT employee\_id, first\_name, last\_name, job\_id, salary

FROM employees

WHERE job\_id = 'IT\_PROG';

COMMIT;

DECLARE

CURSOR c\_employees IS

SELECT \*

FROM employees

WHERE job\_id = 'IT\_PROG';

r\_employees c\_employees%ROWTYPE;

BEGIN

DBMS\_FLASHBACK.enable\_at\_time(sysdate - 30 / 1440); -- Posicionando 10 minutos no passado

OPEN c\_employees; --Executou o Cursor. Vai para o ResultSet

DBMS\_FLASHBACK.disable; -- Posicionando de volta ao presente

LOOP

FETCH c\_employees INTO r\_employees;

EXIT WHEN c\_employees%NOTFOUND;

UPDATE employees

SET salary = r\_employees.salary

WHERE employee\_id = r\_employees.employee\_id;

END LOOP;

CLOSE c\_employees;

COMMIT;

END;

## FlashBack Query

Evolução do DBMS\_FLASHBACK

Consulta como estava a tabela em determinado tempo

CURSOR c\_employees IS

SELECT \*

FROM employees as of timestamp (systimestamp - interval '15' minute);

Consultar a tebela 15 minutos atrás

Texto

Descrição gerada automaticamente

SELECT employee\_id, first\_name, last\_name, job\_id, salary

FROM employees

WHERE job\_id = 'FI\_ACCOUNT';

UPDATE employees

SET salary = salary \* 2

WHERE job\_id = 'FI\_ACCOUNT';

SELECT employee\_id, first\_name, last\_name, job\_id, salary

FROM employees

WHERE job\_id = 'FI\_ACCOUNT';

COMMIT;

DECLARE

CURSOR c\_employees IS

SELECT \*

FROM employees as of timestamp (systimestamp - interval '15' minute);

r\_employees c\_employees%ROWTYPE;

BEGIN

OPEN c\_employees;

LOOP

FETCH c\_employees INTO r\_employees;

EXIT WHEN c\_employees%NOTFOUND;

UPDATE employees

SET salary = r\_employees.salary

WHERE employee\_id = r\_employees.employee\_id;

END LOOP;

CLOSE c\_employees;

COMMIT;

END;

SELECT \* FROM V$PARAMETER WHERE NAME LIKE '%undo%'

## FlashBack Drop

A lixeira comtém todos os objetos dropados até que: Limpe com PURGE; Restaure com o before DROP;

Quando Recicla, vem com os índices, mas não trás as ForeKeys;

Select \* from User\_RecycleBin

Select \* from RecycleBin

FLASHBACK TABLE nome\_tabela to BEFORE DROP;

PURGE User\_RecycleBin –Exclui da Lixeira

create table emplyees\_copia as select \* from employees;

select\* from emplyees\_copia;

Drop table emplyees\_copia;

FLASHBACK TABLE emplyees\_copia to BEFORE DROP;

## FlashBack Table

É o que usa menos

Colocar a tabela em determinado tempo

CREATE TABLE employees\_copia2

AS

SELECT \*

FROM employees;

GRANT FLASHBACK ON hr.employees\_copia2 TO hr;

ALTER TABLE hr.employees\_copia2 ENABLE ROW MOVEMENT;

SELECT \*

FROM hr.employees\_copia2 ;

DELETE FROM hr.employees\_copia2 ;

COMMIT;

SELECT \*

FROM hr.employees\_copia2 ;

FLASHBACK TABLE hr.employees\_copia2 TO TIMESTAMP systimestamp - interval '5' minute;

SELECT \*

FROM hr.employees\_copia2;

# 43 – Lobs – Large Objects

Para vídeos, Pdfs, Áudio;

Não podem ser analisados as estatísticas;

Não podem colocar índice;

Não podem ser relacionado em um VARRAY(Collection);

Lob Interno -> Fica dentro do banco de dados;

CLOB -> Caracteres. Máximo entre 8TB e 128TB. Com indioma padrão definido no banco de dados

BLOB -> Dados binários. Imagens, sons, vídeos, arquivos. Entre 8TB e 128 TB

NCLOB -> Caracteres com idiomas diferentes. Entre 8TB e 128 TB.

Lob Externo -> Grava apenas o diretório de onde está o arquivo;

BFILE -> Grava o nome do Arquivo e o diretório.

## CLOB

Não se pode colocar NULL para o CLOB, tem que colocar EMPTY\_CLOB().