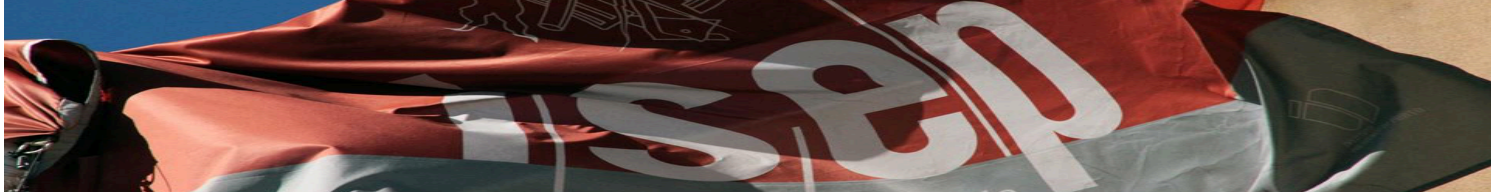


BASE DE DADOS



Oracle
PL/SQL

Teórico-Práticas
Ano Lectivo 2016/2017
Rosa Reis



Objetivos

1. Introdução
2. Blocks
 - *Anonymous Blocks*
 - *Named Blocks*
3. Variáveis e Tipos
4. Sintaxe Básica do PL/SQL
5. Estruturas de Controlo
6. Acedendo aos dados na Base de Dados
7. Cursores
8. Exceções



INTRODUÇÃO

★ PL/SQL

- Extensão ao SQL
- Estruturada em blocos
- Permite controlo do fluxo de execução
- Permite integração entre diferentes ferramentas Oracle
- Não permite comandos DDL

★ PL/SQL combina:

- poder de manipulação de dados do SQL com
- poder de processamento das linguagens procedimentais



BLOCKS

- ★ A unidade básica em PL/SQL é o **Block**. Há vários tipos de blocks:
 - **Anonymous Blocks**: não têm nome (como os *scripts*). Normalmente são construídos dinamicamente e executados apenas uma vez.
 - **Named Blocks**: semelhantes aos anónimos mas com um nome.
 - **Subprogramas**. São *procedures*, *packages*, e *functions* que são guardados na BD. Este blocos normalmente não são alterados depois de construídos e são executados várias vezes. **Os subprogramas são executados explicitamente via uma chamada a um *procedure* ou *function*.**
 - **Triggers**: São named blocks que são também guardados na BD. Este blocos normalmente não são alterados depois de construídos e são executados várias vezes. **Os Triggers são executados implicitamente quando acontecem determinados eventos na BD (Insert, Update e Delete)**



ESTRUTURA DE ANONYMOUS BLOCK

DECLARE

--Definição de objetos PL/SQL a utilizar dentro do bloco.

BEGIN

--Ações executáveis

EXCEPTION

--Processamento de exceções.

END;

- Os blocos podem ser encadeados.
- Os elementos BEGIN e END são obrigatórios e delimitam o conjunto de acções a efectuar.
- A secção **DECLARE** é **opcional** e é utilizada para definir objectos de PL/SQL, tais como as variáveis referenciadas no bloco ou num bloco encadeado.
- **A secção EXCEPTION é opcional** e é utilizada para captar excepções, e definir acções a tomar quando estas ocorrem.
- **Todas as instruções PL/SQL são terminadas com ponto e vírgula.**



EXEMPLO DE ANONYMOUS BLOCK

```
DECLARE --declaration section (types, variables, ...)
```

```
    l_commission                NUMBER;
```

```
    L_COMM_MISSING              EXCEPTION;
```

```
BEGIN --executable section (program body)
```

```
    SELECT commission_pct / 100 INTO l_commission
```

```
    FROM employees WHERE employee_id = emp_id;
```

```
    IF l_commission IS NULL THEN RAISE COMM_MISSING;
```

```
    ELSE      UPDATE employees
```

```
                SET salary = salary + bonus*l_commission
```

```
                WHERE employee_id = emp_id;
```

```
    END IF;
```

```
EXCEPTION --exception section (error handling)
```

```
    WHEN L_COMM_MISSING THEN DBMS_OUTPUT.PUT_LINE('This  
employee does not receive a commission.');
```

```
END;
```




VARIÁVEIS E TIPOS

- ★ Uma variável é um local de armazenamento que pode ser lido ou atribuído pelo programa.
- ★ São declaradas na seção DECLARE dentro de um bloco PL / SQL.
- ★ *Syntax DECLARE*

identifier [CONSTANT] *datatype* [NOT NULL]

[:= | DEFAULT *expr*];

```
Declare
  DataNasc      DATE;
  idade         NUMBER(2) NOT NULL := 30;
  nome          VARCHAR2(50) := 'Daniela';
  Controlador   CONSTANT NUMBER := 77;
  valido        BOOLEAN NOT NULL := TRUE;
```



VARIÁVEIS E TIPOS

★ Escalar

- variavel
- Constante

```
DECLARE  
  l_x NUMBER := 20000;  
  l_message VARCHAR2(40);  
  C_PI CONSTANT NUMBER(3,2) := 3.14;
```

★ Composto/ tipo vector

- Record

```
TYPE T_TIME IS RECORD (minutes INTEGER, hours NUMBER(2));  
current_time_rec T_TIME;  
Current_time_rec.hours := 12;
```




VARIÁVEIS E TIPOS

★ Coleções

➤ Array Associativo

```
DECLARE
    TYPE T_POPULATION IS TABLE OF NUMBER INDEX BY
VARCHAR2 (64) ;
    l_city_population  T_POPULATION;
    l_i number;
BEGIN
    l_city_population('Smallville')  := 2000;
    l_i:= l_city_population('Smallville') ;
END;
/
```



VARIÁVEIS E TIPOS

★ Array de comprimento variavel (VARRAY)

```
DECLARE
  TYPE T_FOURSOME IS VARRAY(4) OF VARCHAR2(15);
  l_team T_FOURSOME := T_FOURSOME('John', 'Mary', 'Alberto');
BEGIN
  l_team.EXTEND;                                -- Append one null element
  l_team(4) := 'Mike';                          -- Set 5th element element
  DBMS_OUTPUT.PUT_LINE( l_team( l_team.first ) ); -- Print first element
  DBMS_OUTPUT.PUT_LINE( l_team( l_team.last ) );  -- Print last element

END;
/
```



VARIÁVEIS E TIPOS

★ Nested Tables

```
DECLARE
    TYPE T_ROSTER IS TABLE OF VARCHAR2(15);
    l_names T_ROSTER := T_ROSTER('D Caruso', 'J Hamil', 'D Piro', 'R Singh');
    l_i number;
BEGIN
    FOR l_i IN l_names.FIRST .. L_names.LAST LOOP    --For first to last element
        DBMS_OUTPUT.PUT_LINE(l_names(l_i));
    END LOOP;
END;
/
```



ATRIBUTOS - %TYPE

★ *Possibilidade de declarar o tipo de dados dependente de um tipo já definido:*

- *Anteriormente no bloco*
- *Numa coluna de uma tabela*

Exemplo:

```
Declare  
  Idade1   number (2) ;  
  Idade2   Idade1%TYPE;  
  Nome     students.name%TYPE;
```



ATRIBUTOS - %ROWTYPE

- ★ *Possibilidade de declarar o tipo de dados com a mesma estrutura do que um tabela*

Exemplo:

```
Declare  
  Estudante      students%ROWTYPE;
```

Para aceder aos valores:

```
Nome :=      Estudante.Name;
```



SINTAXE BÁSICA DO PL/SQL

- ✱ As instruções podem, se necessário, passar de uma linha para a outra, **mas as palavras-chave não podem ser divididas.**
- ✱ As unidades léxicas (identificadores, operadores, etc) podem **ser separadas por um ou mais espaços ou por outros limitadores que não se confundam com a unidade léxica.**
- ✱ Não se podem **usar palavras reservadas como identificadores, excepto se entre aspas.**
- ✱ **Os identificadores** têm que **começar por uma letra e podem ter até 30 caracteres.**
- ✱ Os valores literais do **tipo caracter ou data** têm que ser indicados **entre plicas.**
- ✱ Os literais numéricos podem ser representados por um só valor ou usando notação científica (**2E5=200000**).
- ✱ Os **comentários** podem ser incluídos entre os símbolos **/* e */** quando o comentário engloba várias linhas, ou então **após —** quando o comentário **é apenas uma linha.**



OPERADORES BÁSICOS

+	Adição
-	Subtracção
*	Multiplicação
/	Divisão
IS NULL, LIKE, BETWEEN, IN, =, >, <, <>, !=, ^=, <=, >=	Comparação
:=	Atribuição
	Concatenação
NOT	Negação lógica
AND	Conjunção
OR	Disjunção



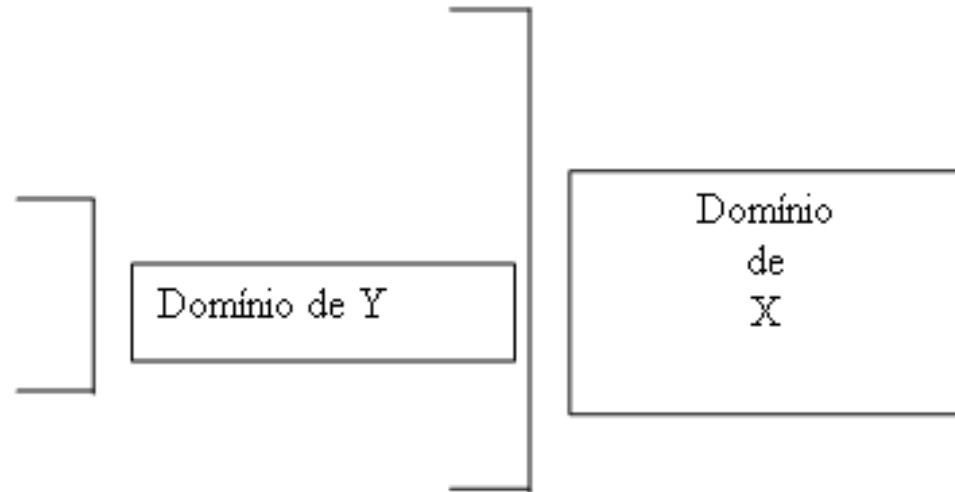
LIMITADORES

(Expressão ou lista
)	Expressão ou lista
;	Fim de instrução
'	Cadeia de caracteres
“	Identificador
<<	Etiqueta
>>	Etiqueta
--	Comentário
/*	Comentário
*/	Comentário



DOMÍNIO DOS OBJETOS

```
DECLARE
    X integer;
BEGIN
    ...
    DECLARE
        Y integer;
    BEGIN
        ...
    END;
    ...
END;
```





ESTRUTURAS DE CONTROLO

IF-THEN-ELSIF

IF condition1 **THEN**

statement1;

ELSIF condition2 **THEN**

statement2;

ELSIF condition3 **THEN**

statement3;

END IF;

```
DECLARE
    l_sales NUMBER(8,2) := 20000;
    l_bonus NUMBER(6,2);
BEGIN
    IF l_sales > 50000 THEN l_bonus := 1500;
        ELSIF l_sales > 35000 THEN l_bonus :=
500;
            ELSE l_bonus := 100;
        END IF;
    UPDATE employees SET salary = salary +
l_bonus;
END;
```



ESTRUTURAS DE CONTROLO

Loops interativos

- Simples loop (infinito)
- WHILE loop
- FOR loop
 - Intervalo numérico
 - Reversed
 - Baseado em Cursores

```
DECLARE
```

```
    l_i          NUMBER := 0;
```

```
BEGIN
```

```
    LOOP
```

```
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(l_i));
```

```
        l_i:=l_i+1;
```

```
    END LOOP;
```

```
    WHILE l_i < 10 LOOP
```

```
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(l_i));
```

```
        l_i := l_i + 1;
```

```
    END LOOP;
```

```
    FOR l_i IN 1..500 LOOP
```

```
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(l_i));
```

```
    END LOOP;
```

```
    FOR l_i IN REVERSE 1..3 LOOP
```

```
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(l_i));
```

```
    END LOOP;
```

```
END;
```



ESTRUTURAS DE CONTROLO

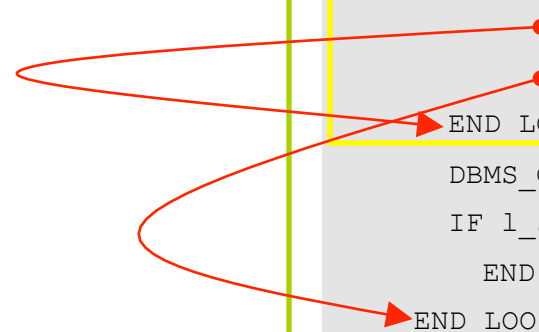
Iterative loops

- Nomeados

Saída do loop

- EXIT statement

```
DECLARE
    l_i NUMBER := 0;
    l_j NUMBER := 0;
    l_s NUMBER := 0;
BEGIN
    <<outer_loop>>
    LOOP
        l_i := l_i + 1;
        <<inner_loop>>
        LOOP
            l_j := l_j + 1;
            l_s := l_s + l_i * l_j;
            EXIT inner_loop WHEN (l_j > 5);
            EXIT outer_loop WHEN ((l_i * l_j) > 15);
        END LOOP inner_loop;
        DBMS_OUTPUT.PUT_LINE('Sum: ' || TO_CHAR(l_s));
        IF l_s > 100 THEN EXIT;
        END IF;
    END LOOP outer_loop;
END;
```





ACESSO A DADOS NA BASE DE DADOS

- ★ Seleccionando no máximo uma linha

- SELECT INTO statement

```
SELECT COUNT(*) INTO variable FROM table;  
SELECT * INTO record FROM table WHERE ...;
```

- ★ Seleccionando várias linhas

- Cursores

- ★ Inserindo e atualizando

```
INSERT INTO table VALUES (var1, var2, ...);
```



CURSORES

- ★ Cada consulta produz um resultado - CURSOR
 - conjunto de linhas que atendem a consulta
 - reside na memória do servidor

```
Select emp_no,emp_name,  
emp_job  
from employees  
where emp_no > 500;
```

Result Set

EMP_NO	EMP_NAME	EMP_JOB	EMP_HIREDATE	EMP_DEPTNO
380	KING	CLERK	1-JAN-1982	10
381	BLAKE	ANALYST	11-JAN-1982	30
392	CLARK	CLERK	1-FEB-1981	30
569	SMITH	CLERK	2-DEC-1980	20
566	JONES	MANAGER	5-JUL-1978	30
788	SCOTT	ANALYST	20-JUL-1981	10
876	ADAMS	CLERK	14-MAR-1980	10
902	FORD	ANALYST	25-SEP-1978	20



DEFININDO CURSORES EXPLICITOS

★ Cursor mais simples:

```
CURSOR my_cursor IS SELECT * from table;
```

★ Sintaxe completa do cursor

```
CURSOR name(parameter_list) RETURN rowtype IS SELECT ...;
```

- O comando select é estático, mas pode ser parametrizado
- O Return é util nos packages

★ Atributos

- %FOUND, %NOTFOUND, %ROWCOUNT, %ISOPEN



USANDO CURSORES EXPLICITOS

★ Resultados obtidos numa consulta colocados num registo

```
DECLARE
    l_employees employees%ROWTYPE;
    CURSOR l_c (p_low NUMBER DEFAULT 0, p_high NUMBER DEFAULT 99) is
        SELECT * FROM employees WHERE job_id > p_low AND job_id < p_high;
BEGIN
    OPEN l_c(3,20);
    LOOP
        FETCH l_c INTO l_employees;
        EXIT WHEN l_c%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(l_employees.last_name ||
            l_employees.job_id );
    END LOOP;
    CLOSE l_c;
END;
```



CURSORES IMPLICITOS

```
DECLARE

    l_rows number(5);

BEGIN

    UPDATE employee SET salary = salary + 1000;

    IF SQL%NOTFOUND THEN

        dbms_output.put_line('None of the salaries where  updated');

    ELSIF SQL%FOUND THEN l_rows := SQL%ROWCOUNT;

        dbms_output.put_line('Salaries for ' || l_rows || ' employees are
updated');

    END IF;

END;
```



CICLOS FOR COM QUERIES

★ Ciclos FOR com subqueries

- Não é necessário declarar o cursor (o cursor é o próprio select).
- Não é possível invocar os atributos de um cursor explícito definido como subquery de um ciclo FOR de cursor (porque não tem nome).

BEGIN

FOR emp_record IN (SELECT ename, deptno FROM emp) LOOP

-- implicit open and implicit fetch occur

IF emp_record.deptno = 30 THEN

...

END LOOP; -- implicit close occurs

END;



EXCEPTIONS

- ★ **Exceção:** condição de erro; quando ocorre o erro é levantada uma exceção que interrompe o fluxo normal de execução do programa e o direciona para uma rotina de tratamento de exceções (***exception handler***)
- ★ Tipos
 - predefinidas
 - definidas pelo utilizador

EXCEPTION

```
WHEN nome_da_exceção  
    THEN relação de comandos;  
WHEN nome_da_exceção  
    THEN relação de comandos;
```



TRATAMENTO DE EXCEÇÕES

- ★ Exceções **predefinidas** são levantadas implicitamente pelo SGBD:
 - **CURSOR_ALREADY_OPEN** -> tentativa de abrir um cursor já aberto
 - **INVALID_CURSOR** -> aceder a um cursor que não está aberto
 - **INVALID_NUMBER** -> conversão inválida de uma string num numero
 - **NO_DATA-FOUND** -> o comando SELECT ... INTO não retornou nenhuma linha
 - **VALUE_ERRORS** -> conversão de tipos sem sucesso ou atribuição de valores superiores à suportada pela variável
 - **TOO_MANY_ROWS** -> comando SELECT ... INTO retornou mais do que uma linha
 - **ZERO_DIVIDE** -> divisão por zero



EXCEÇÕES PREDEFINIDAS

```
DECLARE  
NOME CHAR(15);  
CARGO CHAR(10);  
BEGIN  
    SELECT ENAME, JOB INTO NOME, CARGO  
    FROM EMP  
    WHERE ENAME = 'KONG';
```

```
EXCEPTION  
WHEN NO_DATA_FOUND THEN  
    DBMS_OUTPUT.PUT_LINE('REGISTO INEXISTENTE ' || SYSDATE);  
    WHEN TOO_MANY_ROWS THEN  
    DBMS_OUTPUT.PUT_LINE(' MUITOS REGISTOS ' || SYSDATE);  
    WHEN OTHERS THEN  
    DBMS_OUTPUT.PUT_LINE(' OUTRO ERRO QUALQUER ' || SYSDATE);  
END;
```



EXCEÇÕES DEFINIDAS PELO UTILIZADOR

★ Características

- precisam ser declaradas e chamadas explicitamente

★ Declaração

- realizada na área de declaração
- sintaxe: **nome_da_exceção EXCEPTION**

★ Utilização

- realizada na área de comandos
- sintaxe: **RAISE nome_da_exceção**



EXCEÇÕES DEFINIDAS PELO UTILIZADOR

DECLARE

nome_exceção **EXCEPTION**;

exceção tem que
ser declarada!

BEGIN

relação_de_comandos;

IF ...

THEN **RAISE** nome_exceção;

END IF;

relação_de_comandos;

EXCEPTION

WHEN nome_exceção

THEN relação_de_comandos;

END;



EXCEÇÕES DEFINIDAS PELO UTILIZADOR

```
set serveroutput on;
DECLARE
    nusp aluno.NUSP%Type;
    nome aluno.nome_aluno%Type;
    sexo aluno.sexo_aluno%Type;
    valida_campos EXCEPTION;

BEGIN
    nusp := 4;
    nome := 'Daniel';
    -- sexo := 'm';
    IF (nusp IS NULL) OR (sexo IS NULL) OR (nome IS NULL)
    THEN RAISE valida_campos;
    ELSE INSERT INTO aluno VALUES (nusp, nome, sexo);
    END IF;
```




EXCEÇÕES DEFINIDAS PELO UTILIZADOR

```
EXCEPTION
```

```
    WHEN valida_campos
```

```
    THEN dbms_output.put_line ('Preencher todos os campos.');
```

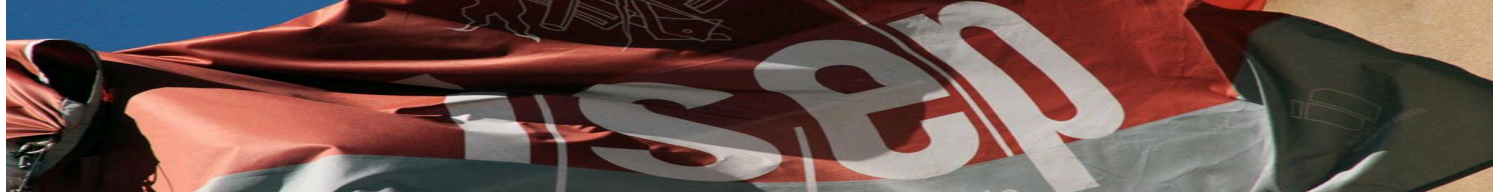
```
    WHEN Dup_Val_On_Index
```

```
    THEN dbms_output.put_line ('Aluno já registado.' );
```

```
    WHEN Others
```

```
    THEN dbms_output.put_line ('Erro no registo.' );
```

```
END;
```



EXERCÍCIO

Crie um bloco em PL/SQL para apresentar os anos bissextos entre 2000 e 2100. Um ano será bissexto quando for possível dividi-lo por 4, mas não por 100 ou quando for possível dividi-lo por 400.

```
DECLARE
V-ANO NUMBER(4);
BEGIN
    FOR V-ANO IN 2000..2100 LOOP
        IF (MOD(V-ANO, 4) = 0 AND MOD(V-ANO, 100) != 0 ) OR (MOD(V-ANO,400) = 0)
        THEN DBMSOUTPUT.PUT_LINE (V-ANO);
        END IF;
    END LOOP;
END;
/
```



EXERCÍCIO

Crie um bloco em PL/SQL para atualizar a tabela, conforme se segue:

- Produtos de categoria A deverão ser reajustados em 5%;
- Produtos de categoria B deverão ser reajustados em 15%;

DECLARE

Cursor c_produto is select * from produto;

v_produto produto%rowtype;

BEGIN

FOR v-produto in c_produto LOOP

IF v-produto.categoria = 'A' then

Update produto set valor= valor *1.05 where codigo =v-produto.categoria;

ELSEIF v_produto.categoria ='B' THEN

UPDATE produto set valor= valor *1.15 where codigo =v-produto.categoria;

END IF;

END LOOP;

END;

/

PRODUTO

CODIGO	CATEGORIA	VALOR
1001	A	7.55
1002	B	5.95
1003	C	3.45