

# Bases de Dados - PLSQL

Engenharia Biomédica, 3º ano

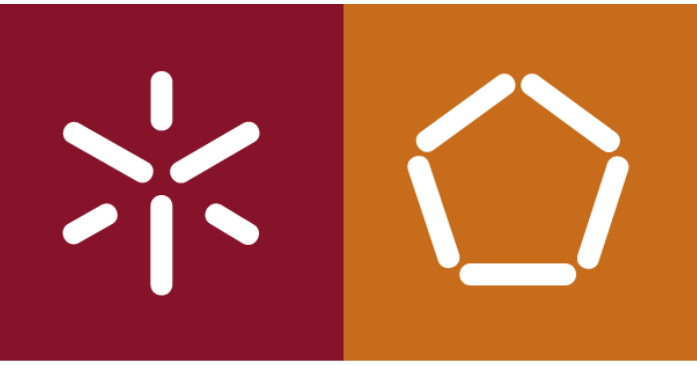
Unidade Curricular de Bases de Dados de Clínica e de Gestão Hospitalar

## PLSQL

António Abelha

Departamento de Informática

Escola de Engenharia

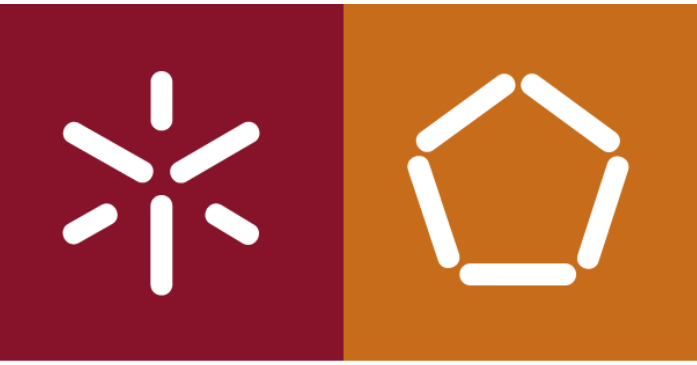


# Bases de Dados - PLSQL

## PL/SQL

- Extensão ao SQL
- Estruturada em blocos
- Permite controlo do fluxo de execução
- Permite integração entre diferentes ferramentas

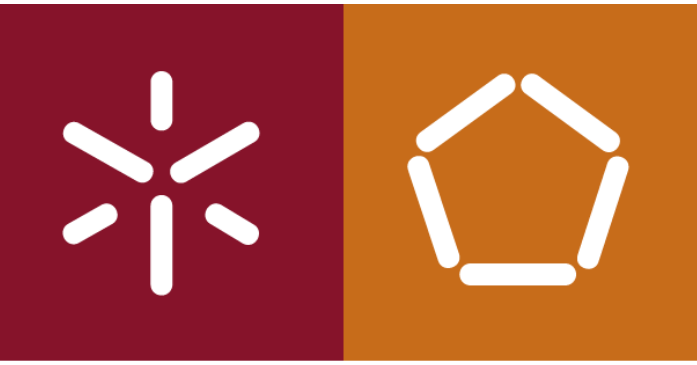
Não permite comandos DDL variáveis e não seguem nenhum esquema predefinido



# Bases de Dados - PL/SQL

## ***PL/SQL combina:***

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



# Bases de Dados - PLSQL

## ***Principais características:***

- Variáveis e constantes
- Tipos de dados escalares e estruturados
- Controlo do fluxo de execução
- Funções integradas
- Gestão de cursores
- Processamento de exceções

Código armazenado na base de dados

# Bases de Dados - PLSQL

## • *Anonymous Blocks*

São blocos anónimos que são declarados numa aplicação no local onde devem ser executados, sendo passados em run-time ao interpretador PL/SQL para execução.

Estruturada em **blocos** (unidade lógica, corresponde a um problema ou sub-problema).

*DECLARE*

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

*BEGIN*

*--Acções executáveis*

*EXCEPTION*

*--Processamento de excepções.*

*END;*

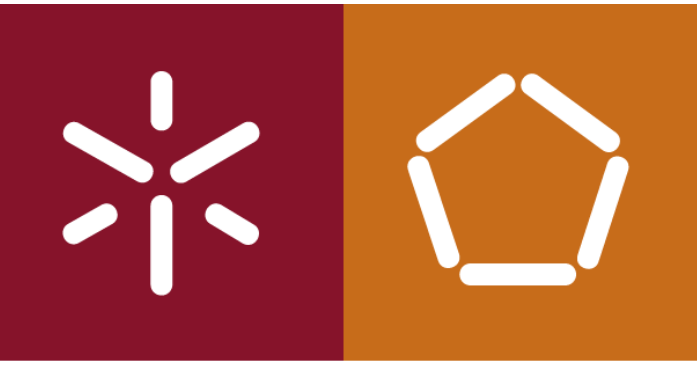
Os blocos podem ser encadeados.

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.



# Bases de Dados - PLSQL

## *Subprograms*

Blocos anónimos com um nome. Podem ser procedimentos ou funções.



# Bases de Dados - PLSQL

## ***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 character 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 comentário podem ser incluídos entre os símbolos /\* e \*/ quando o comentário engloba várias linhas, ou então após – quando o comentário é apenas uma linha.

# Bases de Dados - PLSQL

## Operadores

+ - Adição

- - Subtração / negação

\* - Multiplicação

/ - Divisão

IS NULL, LIKE, BETWEEN, IN, =, >, <, <>, !=, ^=, <=, >= - Comparação

\*\* - Exponenciação

:= - Atribuição

=> - Associação

.. - Intervalo

|| - Concatenação

NOT - Negação lógica

AND - Conjunção

OR - Disjunção



# Bases de Dados - PLSQL

## Limitadores

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

## Separadores

, Itens

## Selectores

. Componente

# Bases de Dados - PLSQL

## ***Declaração de variáveis e constantes***

### Variáveis

*identificador tipo\_de\_dados [(precisão, escala)] [NOT NULL] [:= expressão];*

### Constantes

*identificador CONSTANT tipo\_de\_dados [(precisão, escala)] := expressão;*

### Atribuições

*identificador := expressão;*

# Bases de Dados - PLSQL

## *Domínio dos objectos*

*DECLARE*

*X integer;*

*BEGIN*

*...*

*DECLARE*

*Y integer;*

*BEGIN*

*...*

*END;*

*...*

*END;*

# Bases de Dados - PLSQL

## ***SUBPROGRAMAS***

***PROCEDURE***

***FUNCTION***

***TRIGGER***

# Bases de Dados - PLSQL

## ESTRUTURAS DE CONTROLO

### IF-THEN-ELSE

```

IF condition1 THEN
    statement1;
ELSE
    IF condition2 THEN
        statement2;
    ELSE
        IF condition3 THEN
            statement3;
        END IF;
    END IF;
END IF;

```

### IF-THEN-ELSIF

```

IF condition1 THEN
    statement1;
ELSIF condition2 THEN
    statement2;
ELSIF condition3 THEN
    statement3;
END IF;

```

### LOOP

```

LOOP
    sequence_of_statements;
END LOOP;

```

# Bases de Dados - PLSQL

## ESTRUTURAS DE CONTROLO

### LOOP LABELS

```

<<outer>>
LOOP
  ...
  LOOP
    ...
    EXIT outer WHEN ... -- exit both loops
  END LOOP;
  ...
END LOOP outer;

```

### EXIT

```

LOOP
  ...
  IF count > 100 THEN
    EXIT;
  END IF;
  ...
END LOOP;

```

### EXIT-WHEN

```

LOOP
  ...
  EXIT WHEN count > 100;
  ...
END LOOP;

```



# Bases de Dados - PLSQL

## *ESTRUTURAS DE CONTROLO*

### **WHILE-LOOP**

```
WHILE condition LOOP
    sequence_of_statements;
END LOOP;
```

### **FOR-LOOP**

```
FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
    sequence_of_statements;
END LOOP;
```

# Bases de Dados - PLSQL

## ***CURSORES***

```
CURSOR cursor_name [(parameter[, parameter]...)] IS select_statement;
```

```
cursor_parameter_name [IN] datatype [{:= | DEFAULT} expr]
```

```
DECLARE
  CURSOR c1 IS SELECT ename, job FROM emp WHERE sal < 3000;
  my_record c1%ROWTYPE;
  ...
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO my_record;
    EXIT WHEN c1%NOTFOUND;
    -- process data record
  END LOOP;
  CLOSE c1;
  ...
END;
```

```
DECLARE
  CURSOR c1 (name VARCHAR2, salary NUMBER) IS SELECT ...
BEGIN
  OPEN c1('ATTLEY', 1500);
  ...
END;
```



# Bases de Dados - PLSQL

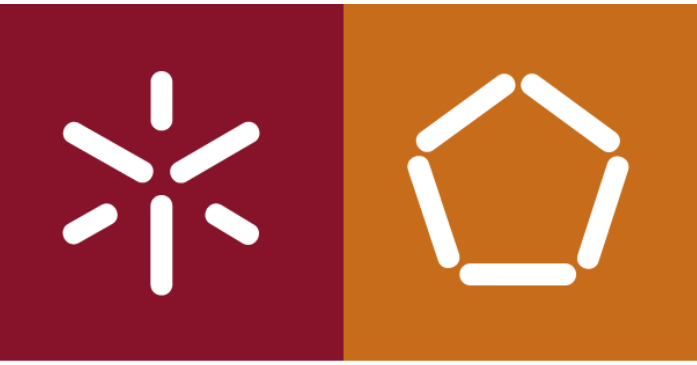
## ***EXCEPÇÕES***

```
DECLARE
    pe_ratio NUMBER(3,1);
BEGIN
    SELECT price / earnings INTO pe_ratio FROM stocks
        WHERE symbol = 'XYZ'; -- pode causar division-by-zero error
        INSERT INTO stats (symbol, ratio) VALUES ('XYZ', pe_ratio);
EXCEPTION
    WHEN ZERO_DIVIDE THEN -- trata 'division by zero' error
        INSERT INTO stats (symbol, ratio) VALUES ('XYZ', NULL);
        ...
    WHEN OTHERS THEN -- handles all other errors
        ...;
END;
```

# Bases de Dados - PLSQL

## *Exemplo de programa:*

```
DECLARE
    qty_on_hand  NUMBER(5);
BEGIN
    SELECT quantity INTO qty_on_hand FROM inventory
        WHERE product = 'TENNIS RACKET'
    IF qty_on_hand > 0 THEN  -- check quantity
        UPDATE inventory SET quantity = quantity - 1
            WHERE product = 'TENNIS RACKET';
        INSERT INTO purchase_record
            VALUES ('Tennis racket purchased', SYSDATE);
    ELSE
        INSERT INTO purchase_record
            VALUES ('Out of tennis rackets', SYSDATE);
    END IF;
COMMIT
```



# Bases de Dados - PLSQL

## ***Tipos de dados:***

### ***VARCHAR(n)***

Conjunto de caracteres (string) de tamanho variável.  $n$  varia entre 1 e 2000 caracteres.

### ***VARCHAR2(n)***

Conjunto de caracteres (string) de tamanho variável.  $n$  varia entre 1 e 4000 caracteres.

### ***NUMBER(p, e)***

Representa um número com uma precisão de  $p$  e uma escala de  $e$ .

### **LONG**

Conjunto de caracteres de tamanho variável até 2 gigabytes.

### **BOOLEAN**

Valor binário



# Bases de Dados - PLSQL

## *Tipos de dados:*

### **DATE**

Data

### **CHAR(*n*)**

Conjunto de caracteres de tamanho fixo. *n* máximo é de 255 bytes e o comprimento por omissão é de 1 byte

### **BLOB, CLOB, NCLOB e BFILE**

tipos de dados para conteúdos binários até 4 Gigabytes internos ou externos (BFILE) à base de dados.

### **RAW(*n*)**

Dados binários em bruto de comprimento variável. *n* máximo é de 255 bytes.

### **LONG RAW**

Dados binários em bruto com um comprimento variável e de tamanho máximo igual a 2 gigabytes.

### **ROWID**

String hexadecimal que representa o endereço único de uma linha numa tabela.

# Bases de Dados - PLSQL

## *Tipos de dados definidos pelo utilizador:*

```
DECLARE
```

```
TYPE TimeRec IS RECORD (minutes SMALLINT, hours SMALLINT) ;
```

```
TYPE MeetingTyp IS RECORD (
```

```
    day      DATE,
```

```
    time     TimeRec,  -- nested record
```

```
    place    VARCHAR2(20),
```

```
    purpose  VARCHAR2(50)) ;
```

# Bases de Dados - PLSQL

## *Declarar variáveis:*

```
part_no  NUMBER(4);  
in_stock BOOLEAN;
```

## *%TYPE e %ROWTYPE:*

```
alunos aluno%ROWTYPE;  
nomeal4 aluno.nome%TYPE;
```

## *Declarar constantes:*

```
credit_limit CONSTANT REAL := 5000.00;
```

# Bases de Dados - PLSQL

## *Instrução de atribuição:*

1. operador :=

```
bonus := current_salary * 0.10;
```

2. atribuir valor com SELECT ou FETCH:

```
SELECT sal * 0.10  
  INTO bonus  
 FROM emp  
WHERE empno = emp_id;
```

# Bases de Dados - PLSQL

## *Declarar cursores:*

```

DECLARE
  CURSOR c1 IS
    SELECT empno, ename, job FROM emp WHERE deptno = 20;

```

Comandos `OPEN`, `FETCH` e `CLOSE` permitem operar o cursor

```

DECLARE
  CURSOR c1 IS SELECT ename, sal, hiredate, job FROM emp;
  emp_rec c1%ROWTYPE;

...
OPEN c1
...
FETCH c1 INTO emp_rec;
...
emp_rec.sal := emp_rec.sal * 1.05 - aumento de 5%

```





# Bases de Dados - PLSQL

**Excepção:** condição de erro; quando ocorre o erro é levantada uma excepção que interrompe o fluxo normal de execução do programa e o direcciona para uma rotina de tratamento de excepções (***exception handler***)

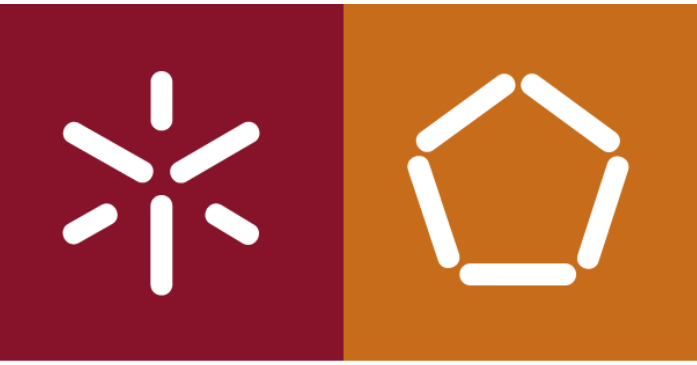
Excepções **pré-definidas** são levantadas implicitamente pelo sgbd:

- **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 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 retornou mais do que uma linha
- **ZERO\_DIVIDE** -> divisão por zero

# Bases de Dados - PLSQL

Excepções **definidas pelo utilizador** têm que ser declaradas e são levantadas com o comando **RAISE**

```
DECLARE
    ...
    comm_missing EXCEPTION; -- declare exception
BEGIN
    ...
    IF commission IS NULL THEN
        RAISE comm_missing; -- raise exception
    ELSE
        bonus := (salary * 0.10) + (commission * 0.15);
    END IF;
EXCEPTION
    WHEN comm_missing THEN
        -- process error      -- exception handler
```



# Bases de Dados - PLSQL

## PROCEDURE

**Para realizar uma determinada acção**

Síntaxe:

```
PROCEDURE name [(parameter[, parameter, ...])] IS
    [local declarations]
BEGIN
    executable statements
[EXCEPTION
    exception handlers]
END [name];
```

Especificação de parâmetros:

```
parameter_name [IN | OUT | IN OUT] datatype [{:= | DEFAULT} expression]
```

# Bases de Dados - PLSQL

## Exemplo:

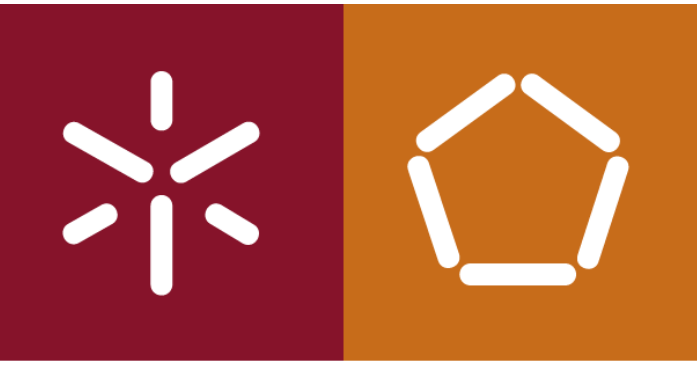
```

PROCEDURE raise_salary (emp_id INTEGER, increase REAL) IS
    current_salary REAL;
    salary_missing EXCEPTION;
BEGIN
    SELECT sal INTO current_salary FROM emp
        WHERE empno = emp_id;
    IF current_salary IS NULL THEN
        RAISE salary_missing;
    ELSE
        UPDATE emp SET sal = sal + increase
            WHERE empno = emp_id;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        INSERT INTO emp_audit VALUES (emp_id, 'No such number');
    WHEN salary_missing THEN
        INSERT INTO emp_audit VALUES (emp_id, 'Salary is null');
END raise_salary;

```

Um procedure é invocado como um comando PL/SQL:

```
raise_salary(emp_num, amount);
```



# Bases de Dados - PLSQL

## FUNCTION

Para calcular e retornar um valor

Síntaxe:

```
FUNCTION name [(parameter[, parameter, ...])] RETURN datatype IS
    [local declarations]
BEGIN
    executable statements
[EXCEPTION
    exception handlers]
END [name];
```

Especificação de parâmetros:

```
parameter_name [IN | OUT | IN OUT] datatype [{:= | DEFAULT} expression]
```

# Bases de Dados - PLSQL

A sintaxe de uma *function* é idêntica à de um *procedure* mas a *function* contém uma cláusula RETURN que especifica o tipo de dados de retorno

**Exemplo:**

```
FUNCTION sal_ok (salary REAL, title REAL) RETURN BOOLEAN IS
  min_sal REAL;
  max_sal REAL;
BEGIN
  SELECT losal, hisal INTO min_sal, max_sal
  FROM sals
  WHERE job = title;
  RETURN (salary >= min_sal) AND (salary <= max_sal);
END sal_ok;
```

Uma *function* é invocada como parte de uma expressão; o identificador de função actua como uma variável cujo valor depende dos parâmetros:

```
IF sal_ok(new_sal, new_title) THEN ...
```

Instrução **RETURN**: termina a execução de um sub-programa e retorna o controlo para o ponto de chamada

Cada sub-programa pode ter várias instruções RETURN (embora seja má prática de programação)

Num PROCEDURE não podem ter uma expressão associada

Numa FUNCTION têm que ter uma expressão associada

Uma FUNCTION tem que ter obrigatoriamente uma instrução RETURN, caso contrário o PL/SQL levanta a excepção PROGRAM\_ERROR



## Bases de Dados - PLSQL

### MODOS/COMPORTAMENTO

**IN** – internamente são como constantes, não podem ser alterados no sub-programa (modo por defeito), passagem por referência

**OUT** – para retornar valores

**IN OUT** – permite a passagem de valores para o sub-programa e o retorno, passagem por valor





# Bases de Dados - PLSQL

## TRANSACÇÕES

### Controlo de concorrência: acessos simultâneos ao mesmo objecto

O acesso simultâneo a dados é controlado com mecanismos de **lock**:

#### tabelas

```
LOCK TABLE emp IN ROW SHARE MODE NOWAIT;
```

#### ou linhas/registos

```
DECLARE
```

```
CURSOR c1 IS SELECT empno, sal FROM emp
```

```
WHERE job = 'SALESMAN' AND comm > sal FOR UPDATE;
```

A primeira instrução de um programa PL/SQL **inicia uma transacção** que decorre até que ocorra um COMMIT ou ROLLBACK; a primeira instrução após COMMIT ou ROLLBACK inicia nova transacção

**COMMIT**: fecha a transacção em curso e torna definitivas as alterações efectuadas sobre a bd durante essa transacção; liberta todos os locks a tabelas e linhas

**ROLLBACK**: fecha a transacção em curso e desfaz as alterações efectuadas sobre a bd durante essa transacção; coloca a bd no estado em que estava antes do início da transacção; liberta todos os locks a tabelas e linhas



# Bases de Dados - PLSQL

**SAVEPOINT:** marca um ponto no código como referência para realizar ROLLBACKs parciais

```
BEGIN
```

```
...
```

```
SAVEPOINT my_point;
```

```
UPDATE emp SET ... WHERE empno = emp_id;
```

```
...
```

```
SAVEPOINT my_point; -- move my_point to current point
```

```
INSERT INTO emp VALUES (emp_id, ...);
```

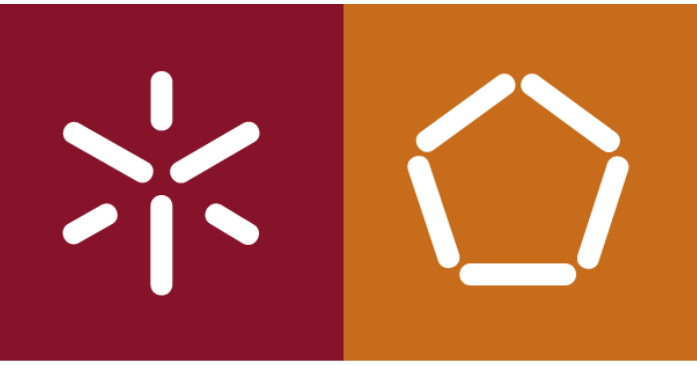
```
...
```

```
EXCEPTION
```

```
WHEN OTHERS THEN
```

```
ROLLBACK TO my_point;
```

```
END;
```



# Bases de Dados - PLSQL

## TRIGGER

**Triggers são procedimentos de PL/SQL que são executados (disparados) quando ocorre um dos seguintes tipos de operações:**

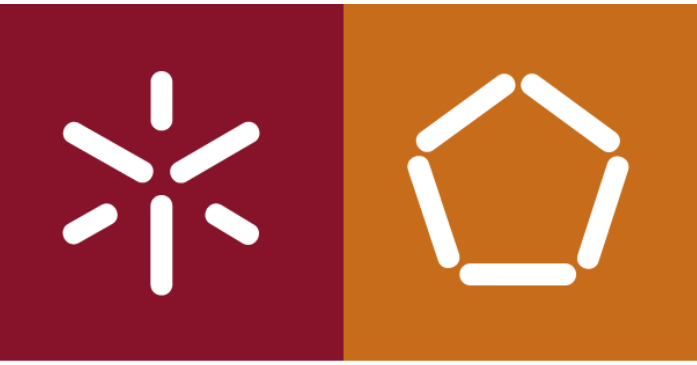
**instruções de DML num objecto schema especifico**

**instruções de DDL feitos num schema ou numa bd**

**eventos de Login/Logoff do utilizador**

**erros de servidor**

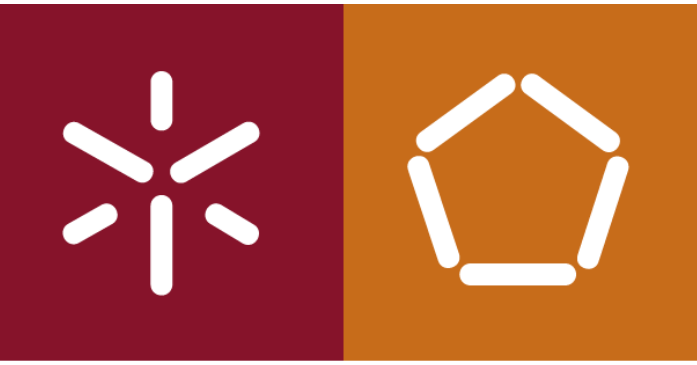
**Startup/Shutdown da bd**



# Bases de Dados - PLSQL

## Principal diferença PROCEDURE/TRIGGER

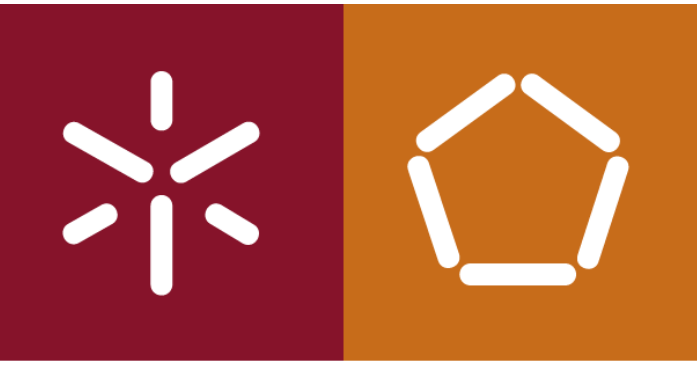
- 1. PROCEDURE é executado quando invocado explicitamente pelo utilizador**
- 2. TRIGGER é executado quando invocado implicitamente pelo SGBDs sempre que um evento de triggering ocorra, independentemente do utilizador ou aplicação que o use**
- 3. A utilização de triggers deve ser muito cuidadosa (apenas quando necessário) o uso excessivo de triggers pode resultar em interdependências complexas (*Cascading Triggers*) que dificultam a manutenção de grandes aplicações.**



# Bases de Dados - PLSQL

## Utilização de triggers na restrição de dados de input:

- 1 - quando uma regra de integridade não pode ser assegurada através de:
  - NOT NULL, UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK
  - DELETE CASCADE
  - DELETE SET NULL
- 2 - para assegurar regras de negócio complexas que não é possível impôr através de CONSTRAINTS
- 3 - para assegurar a integridade referencial quando tabelas “filho” e tabelas” pai” estão em diferentes nós de uma bd distribuida



# Bases de Dados - PLSQL

**Um Trigger é composto por 3 partes:**

- O evento ou instrução de Triggering;**
- A restrição;**
- A acção ou corpo;**

**Quando se define um Trigger é possível especificar se este deve ser executado:**

- para cada linha afectada pela instrução de triggering, tal como um Update statement que actualiza 'n' linhas. (triggers de linha)**
- para cada instrução de triggering, independentemente do numero de linhas que afecte (triggers de instrução)**
- antes da instrução de triggering**
- depois da instrução de triggering**

# Bases de Dados - PLSQL

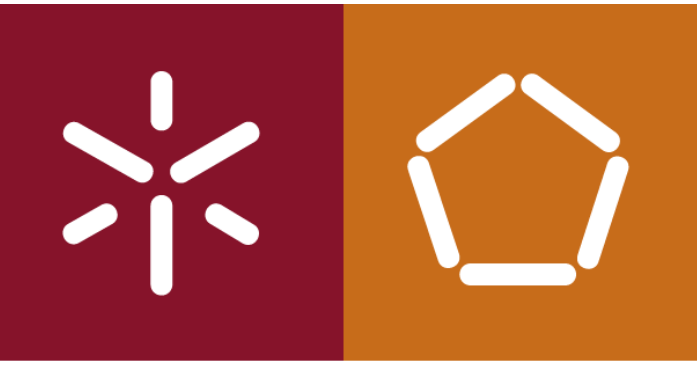
## Exemplo de criação de um trigger:

```

CREATE OR REPLACE TRIGGER Trg_Mostra_Encomenda -- nome do trigger
BEFORE INSERT OR UPDATE ON VendasDetalhes -- instrução de Triggering
FOR EACH ROW
WHEN (new.qtd_encomenda > 0) -- restrição
DECLARE - inicio da acção ou corpo do trigger
v_dif number;
BEGIN
v_dif := :new.qtd_encomenda - :new.qtd_enviada;
dbms_output.put_line ('Trigger TRG_MOSTRA_ENCROMENDA disparou!');
dbms_output.put_line('Quantidade encomendada: ' || :new.qtd_encomenda)
dbms_output.put_line('Quantidade Enviada: ' || :new.qtd_enviada)
dbms_output.put_line(' Quantidade por enviar: ' || v_dif);
END;

```





## Bases de Dados - PLSQL

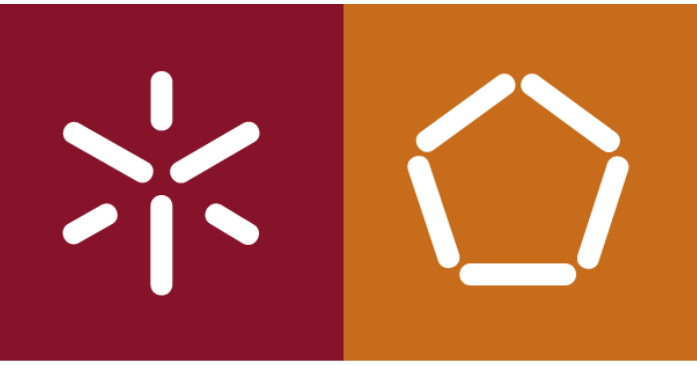
### o Timming dum Trigger

**A opção BEFORE ou AFTER no CREATE TRIGGER especifica se a acção do trigger deve ser executada ANTES ou DEPOIS da instrução de triggering a ser executada.**

### Executando o trigger uma ou mais vezes (FOR EACH ROW)

**Esta opção, quando especificada, "dispara" o trigger em cada registo afectado pela instrução de triggering.**

**A ausência desta opção indica que o trigger só é executado uma única vez para cada instrução e não separadamente para cada registo afectado**



# Bases de Dados - PLSQL

## Tipo Trigger

### Características

#### **BEFORE instrução**

A acção do trigger é executada antes da instrução de triggering;

#### **AFTER instrução**

A acção do trigger é executada depois de executada a instrução de triggering

#### **BEFORE linha**

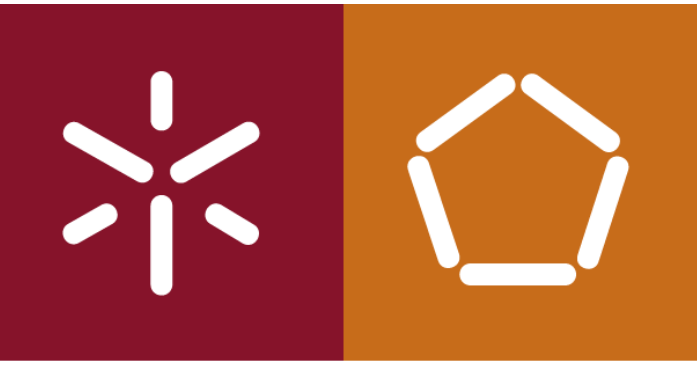
A acção do trigger é executada:

- de acordo com a restrição do trigger;
- antes de cada linha ser afectada pela instrução de triggering;
- antes da verificação das restrições de integridade.

#### **AFTER linha**

A acção do trigger é executada para cada registo de acordo com a restrição do trigger e depois de modificados os registos pela instrução de triggering. É feito o lock dos registos afectados.





## Bases de Dados - PLSQL

Vários triggers do mesmo tipo para a mesma tabela.

**Aceder aos valores dos campos nos TRIGGER DE LINHAS**

**Nomes de correlação**

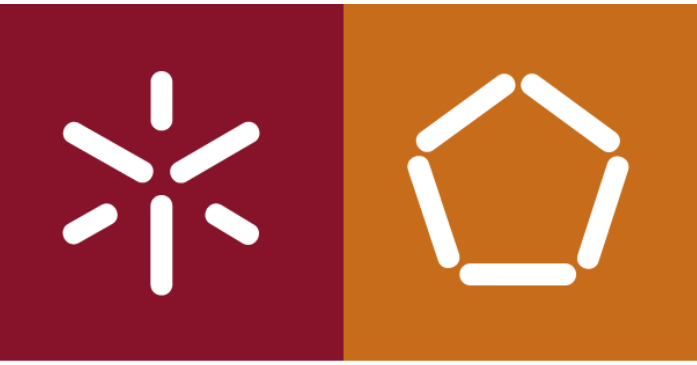
No corpo dum trigger é possível aceder aos valores antigos e novos dos campos do registo afectado pela instrução de triggering.

Existem dois nomes de correlação para cada coluna da tabela a ser modificada:

- um para o valor antigo ( **:OLD** ) e
- outro para o valor novo ( **:NEW** ):

# Bases de Dados - PLSQL

| <b>Operação</b> | <b>:OLD (utilidade)</b> | <b>:NEW (utilidade)</b> |
|-----------------|-------------------------|-------------------------|
| <b>INSERT</b>   | <b>X</b>                | √                       |
| <b>UPDATE</b>   | √                       | √                       |
| <b>DELETE</b>   | √                       | <b>X</b>                |



# Bases de Dados - PLSQL

## Detectar a operação DML que “disparou” o trigger (INSERTING, UPDATING, e DELETING)

Mais do que um tipo de operação DML pode disparar um trigger (por exemplo: ON INSERT OR DELETE OR UPDATE of Editoras).

Na acção do trigger utilizam-se predicados condicionais ( INSERTING, DELETING e UPDATING) para verificar qual dos tipos de operação “disparou” o trigger.

```
IF INSERTING THEN ... END IF; -- TRUE se foi um INSERT que “disparou” o trigger
```

```
IF UPDATING THEN ... END IF; -- TRUE se foi um UPDATE que “disparou” o trigger
```

Num trigger de UPDATE pode-se especificar o nome do campo a ser actualizado.

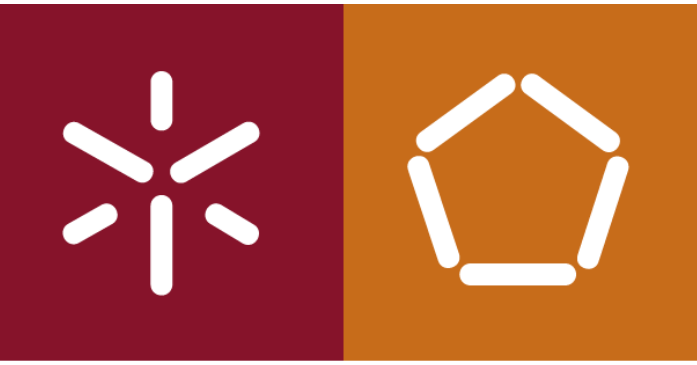
```
CREATE OR REPLACE TRIGGER ...
```

```
... UPDATE OF qtd_encomenda, qtd_enviada ON VendasDetalhes ...
```

```
BEGIN
```

```
... IF UPDATING ('qtd_encomenda') THEN ... END IF;
```

```
END;
```



# Bases de Dados - PLSQL

## Exceções em Triggers

Tal como nos sub-programas, também pode ser feito o tratamento de exceções em triggers.

```
CREATE OR REPLACE
TRIGGER trg_salarios2
BEFORE UPDATE OF salary ON employee
FOR EACH ROW
  DECLARE
    too_much EXCEPTION;
BEGIN
  IF :NEW.salary>99000 THEN
    RAISE too_much;
  END IF;
  EXCEPTION
    WHEN too_much THEN
      RAISE_APPLICATION_ERROR (-20001, 'Cannot pay that much');
END;
```



# Bases de Dados - PLSQL

- ◆ Connolly, T., Begg, C., **Database Systems, A Practical Approach to Design, Implementation, and Management** , Addison-Wesley, 6a Edição, 2014.
- ◆ Luc Perkins, Eric Redmond, Jim Wilson, **Seven Databases in Seven Weeks - A Guide to Modern Databases and the NoSQL Movement**, Pragmatic Bookshelf, 2018.