



Introdução ao PL/SQL

# AULA PL05

Hugo Peixoto

2019 – 2020 Universidade do Minho



- Views
  - Criação
- Procedures
  - Criação e utilização
- Functions
  - Criação e utilização
- Sequences
  - Criação e utilização
- Triggers
  - Criação



# Views

**Base  
Table**

employees						
employee_id	last_name	job_id	manager_id	hire_date	salary	department_id
203	marvis	hr_rep	101	07-Jun-94	6500	40
204	baer	pr_rep	101	07-Jun-94	10000	70
205	higgins	ac_rep	101	07-Jun-94	12000	110
206	gietz	ac_account	205	07-Jun-94	8300	110

**View**

staff				
employee_id	last_name	job_id	manager_id	department_id
203	marvis	hr_rep	101	40
204	baer	pr_rep	101	70
205	higgins	ac_rep	101	110
206	gietz	ac_account	205	110



# Views

```
create table EMPLOYEES (  
    employee_id number primary key,  
    last_name varchar2(50) not null,  
    job_id varchar2(50) not null,  
    manager_id  number not null,  
    hire_date date not null,  
    salary number not null,  
    department_id number not null  
);
```



# Views

```
insert into employees values (203, 'marvis', 'hp_rep',101,to_date('07-06-2004',  
'dd-mm-yyyy'), 6500, 40);
```

```
insert into employees values (204, 'baer', 'pr_rep',101,to_date('01-06-2004',  
'dd-mm-yyyy'), 10000, 70);
```

```
insert into employees values (205, 'higgins', 'ac_rep',101,to_date('21-06-2004',  
'dd-mm-yyyy'), 12000, 110);
```

```
insert into employees values (206, 'gietz', 'ac_account',101,to_date('24-06-  
2004', 'dd-mm-yyyy'), 8300, 110);
```

```
insert into employees values (207, 'john', 'hp_rep',205,to_date('12-06-2004',  
'dd-mm-yyyy'), 6500, 40);
```

```
create or replace view staff as  
    select employee_id, last_name, job_id, manager_id,  
    department_id from employees;
```



# Procedures and Functions

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms:

- **Procedures** – These subprograms do not return a value directly; mainly used to perform an action.
- **Functions** – These subprograms return a single value; mainly used to compute and return a value.



# Procedures and Functions

Parts & Description
<b>Declarative Part</b> It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.
<b>Executable Part</b> This is a mandatory part and contains statements that perform the designated action.
<b>Exception-handling</b> This is again an optional part. It contains the code that handles run-time errors.



# Procedures

## Creating a Procedure

```
CREATE [OR REPLACE] PROCEDURE procedure_name  
[(parameter_name [IN | OUT | IN OUT] type [,  
...])]  
{IS | AS}  
BEGIN  
    < procedure_body >  
END procedure_name;
```





# Procedures

Where:

[OR REPLACE] option allows the modification of an existing procedure.

`procedure_name` specifies the name of the procedure.

The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.

`procedure-body` contains the executable part.



# Procedures

```
CREATE OR REPLACE PROCEDURE greetings AS
    BEGIN
        dbms_output.put_line('Hello World!');
    END;

call greetings;
```



# Procedures

```
create table cars (ide number primary key, nome varchar2(10), valor number);
```

```
insert into cars values (1,'Ford',10);
```

```
insert into cars values (2,'VW',20);
```

```
insert into cars values (3,'BMW',30);
```

```
create or replace procedure act_val(ident number, val number) as  
begin
```

```
    update cars c1
```

```
        set valor = nvl(val + (select valor from cars c2  
where c1.ide = c2.ide), valor)
```

```
        where c1.ide = ident;
```

```
end;
```



# Procedures

Testar o uso do procedure

Listar todos os carros:

```
select * from cars;
```

IDE	NOME	VALOR
1	Ford	10
2	VW	20
3	BMW	30

Executar o procedure:

```
call act_val(3,50);
```

Listar novamente todos os carros:

```
select * from cars;
```

IDE	NOME	VALOR
1	Ford	10
2	VW	20
3	BMW	80



# Functions

A stored function (also called a user function or user-defined function) is a set of PL/SQL statements you can call by name.

Stored functions are very similar to procedures, except that a function returns a value to the environment in which it is called. User functions can be used as part of a SQL expression.



# Functions

```
CREATE [OR REPLACE] FUNCTION function_name  
[(parameter_name [IN | OUT | IN OUT] type [, ...])]  
RETURN return_datatype  
{IS | AS}  
BEGIN  
    < function_body >  
END [function_name];
```



# Functions - example

```
create table customers (ide number primary key, nome varchar2(10));
```

```
insert into customers values (1,'a');
```

```
insert into customers values (2,'b');
```

```
insert into customers values (3,'c');
```

```
create or replace function customerName(ident number)
    return varchar2 as
        nome varchar2(10);
begin
        select nome into nome from customers where
customers.ide=ident;
        return nome;
end;
```



# Functions - example

Testar o uso da função:

Verificar qual o nome cujo id= 1:

```
select customerName(1) from dual;
```

CUSTOMERNAME(1)
a

Listar todos cujo nome seja igual ao nome do customerName(1):

```
insert into customers values (4,'a');
```

IDC	NOME
1	a
4	a

```
select * from customers c1 where customerName(1) = c1.nome;
```





# Sequences

Sequence numbers are Oracle integers of up to 38 digits defined in the database.

A sequence definition indicates general information, such as the following:

- The name of the sequence

- Whether the sequence ascends or descends

- The interval between numbers

- Whether Oracle should cache sets of generated sequence numbers in memory



# Sequences

Create:

```
create sequence my_sequence start with 1;
```

CURRVAL returns the current value from sequence:

```
select my_sequence.CURRVAL from dual;
```

NEXTVAL increments the sequence and returns the new value:

```
select my_sequence.NEXTVAL from dual;
```



# Triggers

Triggers are executed on {INSERT, DELETE and UPDATE} and {BEFORE, AFTER} those actions.

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
  AFTER INSERT
    ON table_name
      [ FOR EACH ROW ]
  DECLARE
    -- variable declarations
  BEGIN
    -- trigger code
  EXCEPTION
  WHEN ...
    -- exception handling
END;
```



# Triggers

FOR EACH ROW, o trigger is row-level; otherwise statement-level.

Row-level triggers::

{ Variables NEW e OLD are available to refer to the field before and after the transactions }

In the trigger body, NEW e OLD must be preceeded by ":",  
That is not the case in the WHEN clause.

- REFERENCING: used to make aliases to the NEW, OLD variables.
- Restrictions can be specified in the WHEN clause. This clause can contain subqueries.



# Triggers

```
CREATE TABLE T4 (a INTEGER, b CHAR(10));  
CREATE TABLE T5 (c CHAR(10), d INTEGER);
```

```
CREATE TRIGGER trig1  
  AFTER INSERT ON T4  
  REFERENCING NEW AS newRow  
  FOR EACH ROW  
    WHEN (newRow.a <= 10)  
  BEGIN  
    INSERT INTO T5 VALUES(:newRow.b, :newRow.a);  
  END;
```



# Triggers

```
CREATE TABLE T1 (a INTEGER, b CHAR(10));  
CREATE TABLE T2 (c CHAR(10), d INTEGER);  
  
CREATE TRIGGER trig2  
  AFTER INSERT ON T1  
  FOR EACH ROW  
    WHEN (new.a <= 10)  
  BEGIN  
    INSERT INTO T2 VALUES(:new.b, :new.a);  
  END;
```



# Triggers

The example below creates a table and uses a trigger to populate the primary key :

```
create sequence simple_employees_seq start with 10 increment by 10;

create table SIMPLE_EMPLOYEES (
    empno number primary key,
    name varchar2(50) not null,
    job varchar2(50)
);

create or replace trigger SIMPLE_EMPLOYEES_BIU_TRIG
before insert on SIMPLE_EMPLOYEES
for each row
begin
    if inserting and :new.empno is null
        then :new.empno := simple_employees_seq.nextval;
    end if;
end;
```



# Triggers

Test sequence and trigger:

```
insert into simple_employees (name, job) values ('Mike', 'Programmer');  
insert into simple_employees (name, job) values ('Taj', 'Analyst');  
insert into simple_employees (name, job) values ('Jill', 'Finance');  
insert into simple_employees (name, job) values ('Fred', 'Facilities');  
insert into simple_employees (empono, name, job) values (null, 'Sabra',  
'Programmer');
```

1

```
select empno, name, job from simple_employees order by empno;
```





PL/SQL Tutorial:

<https://www.tutorialspoint.com/plsql/index.htm>



Introdução ao PL/SQL

# AULA PL05

Hugo Peixoto

2019 – 2020 Universidade do Minho