

PL/SQL e Unidades de Programas

Marilde Santos

Índice

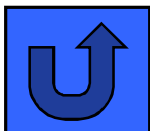
- PL/SQL
 - Processamento Condicional:
 - If
 - Case
 - Processamento Repetitivo
 - For
 - While
 - Loop
 - Forall
- Unidades de Programa
- Cursor

PL/SQL

- Linguagem de Programação **P**rocedural **L**anguage / **S**tructured **Q**uery **L**anguage une o estilo modular de linguagens de programação à versatilidade no acesso a banco de dados obtidas via SQL.

PL/SQL ou Java?

- PL/SQL é proprietária da Oracle, assim, caso seja necessário migrar para outro SGBD, perde-se todo o trabalho em termos de Stored Procedures, Triggers e Functions.
- Alternativa: usar a linguagem Java.
 - Para saber mais: ORACLE9I JDBC PROGRAMMING Nirva Morisseau-Leroy e outros Ed. Oracle Press - 2001



PL: Processamento condicional

If *condição1* then

Comandos executados caso a condição1 seja verdadeira

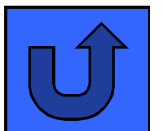
[Elseif *condição2*

Comandos executados caso a condição2 seja verdadeira]

[Else

Comandos executados caso nenhuma condição seja verdadeira]

End if;



PL: Processamento condicional

Case

When *condição* (*atributo op relacional valor*)

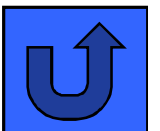
then *valor que o atributo assume se a condição for verdadeira*

When *condição*

then *valor que o atributo assume se a condição for verdadeira*

Else *valor que o atributo assume se nenhuma condição anterior for verdadeira;*

End;



PL: Processamento Repetitivo

- FOR: repete n vezes com n conhecido

FOR I in 1..max LOOP

comandos que devem ser repetidos

END LOOP;

Obs.: as variáveis que controlam o número de repetições (*I*) não precisam ser declaradas nem incrementadas.

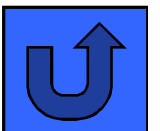
PL: Processamento Repetitivo

- FOR: pode ter contagem regressiva

FOR I in REVERSE max..1 LOOP

comandos que devem ser repetidos

END LOOP;



PL: Processamento Repetitivo

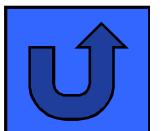
- WHILE: efetua a iteração mediante teste.

WHILE condição LOOP

comandos que devem ser repetidos

END LOOP;

Obs: as variáveis que controlam a iteração devem ser declaradas e explicitamente incrementadas.



PL: Processamento Repetitivo

- LOOP: repete infinita vezes até que seja explicitamente forçado o fim do laço.

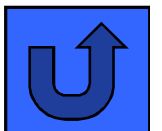
LOOP

comandos que devem ser repetidos

EXIT WHEN *condição*;

END LOOP;

Obs: as variáveis que controlam a iteração devem ser declaradas e explicitamente incrementadas.



PL: Processamento Repetitivo

- FORALL: implementa a técnica **bulk binds**, que consiste em pré-armazenar um conjunto de comandos DML e envia-los de uma vez ao núcleo SQL.

FORALL j in 1..Max

*comando (insert, update ou delete) a repetir; **

** Admite um único comando por vez!*

PL: Processamento Repetitivo

create or replace procedure Alimenta_Historico_Forall

(ultima_turma in number, ultimo_aluno in number)

is

type tlista is table of number index by binary_integer;

lista tlista;

begin

for j in 1..100 loop

lista(j) := j;

end loop;

delete historico;

for i in 1..ultima_turma loop

forall j in 1..ultimo_aluno

insert into historico (cod_turma, matricula) values (i,lista(j));

end loop;

commit;

END;

/

Alimenta-se uma
variável composta!

Para depois usa-la
no insert!



Unidades de Programa

- Um bloco possui a seguinte estrutura:

[declare] *// declaração de variáveis, constantes e cursores. Contém inicializações.*

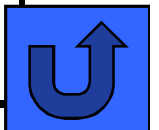
Begin *// comandos SQL e estruturas de programação (if, while, etc)*

[Exception] *// identificação dos erros e emissão de mensagens*

End;

Tipos de Unidades de Programa

<u>Procedure</u>	Pode receber parâmetros de entrada ou de saída. Ativado como se fosse um comando da linguagem.
<u>Function</u>	Pode receber parâmetros apenas de entrada e, necessariamente, retorna um valor em seu nome. A ativação ocorre em expressões.
Package	Reunião física de procedures, functions e cursores.
<u>Trigger</u>	Rotina disparada automaticamente antes ou depois de comandos update, insert ou delete.



Procedure

- Pequenas porções de código que realizam tarefas específicas e ativadas como comandos
- Podem receber parâmetros de:
 - Entrada (In)
 - Saída (Out)
 - Entrada e Saída (InOut)

Procedure

- Sintaxe:

CREATE [OR REPLACE] PROCEDURE

nome_procedure

(*[lista de parâmetros]*)

IS

declarações locais

BEGIN

comandos

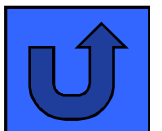
END;

Procedure

```
create or replace procedure AlimentaHistorico  
    (ultima_turma in number, ultimo_aluno in number)  
is  
begin  
    delete historico;  /* comentário: elimina registros atuais */  
    for i in 1..ultima_turma loop  
        for j in 1..ultimo_aluno loop  
            insert into historico (cod_turma, matricula) values (i,j);  
        end loop;  
    end loop;  
    commit;  
end;  
/
```

Execução: **exec** alimentahistorico(10,10);

Verificando a existência: **Select** object_name **from** user_objects
 where object_type='PROCEDURE';



Function

- Podem receber apenas parâmetros de entrada e devolvem um valor em seu nome.
- Sintaxe:

CREATE [OR REPLACE] FUNCTION

Nome_função ([lista de parâmetros])

RETURN tipo de retorno

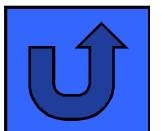
IS

declarações locais

BEGIN

comandos

END;



Function

create or replace function ValorEmDolar

(reais **in** number, cotacao **in** number)

return number

is

begin

return reais/cotacao;

end;

/

Execução:

Select nome_curso, preco “Em R\$”, **ValorEmDolar(preco, 2.97)** “Em
US\$”

From cursos;

Verificando a existência: **Select** object_name **from** user_objects
where object_type='FUNCTION';

Triggers

Triggers representam unidades de programa que são executadas, automaticamente, antes ou depois de um comando disparador, que pode ser tanto um DML (update, insert ou delete) como um DDL (create, alter, drop, truncate table), ou mesmo um evento ocorrido no BD (conexão, por exemplo).

Pra que serve?

- Preenchimento de campo resultante de uma expressão;
- Crítica aos dados com mensagens mais adequadas às regras do negócio;
- Acessos que alterem linhas de uma tabela ou eventos que ocorram no BD podem ser registrados em outra tabela (auditoria);

Pra que serve?

- Permite alterações (insert, update, delete) sobre views que normalmente não poderiam ser modificadas (possuindo cláusula group by, distinct ou operador union, por exemplo);
- Acompanhar o que ocorre após eventos no BD (conexões, erros, etc.)

Triggers X Procedures

Trigger	Procedure ou Function
Ativado implicitamente	Ativado explicitamente
Proibidos: commit, rollback ou savepoint	Esses comandos são permitidos
Quem ativa não precisa possuir privilégio de execução	Quem ativa precisa possuir privilégio de execução
Não se pode emitir um comando select/into em tabelas afetadas pelo Trigger.	Restrição não se aplica

Tipos de Trigger

- **DML: Disparado a partir de um um insert, update ou delete**
- **Instead Of: inserções, deleções ou alterações em views**
- **Schema: disparado a um comando DDL**
- **Database: executado a partir de um evento ocorrido no âmbito do BD.**

Triggers Instead of

-- Exemplo: Alterando dados através de uma View com UNION

-- criação da view:

create view vPessoas as

select nome_aluno as nome, 'a' as tipo

from alunos

union

select nome_instrutor, 'i'

from instrutores;

Triggers Instead of

-- criação do trigger:

create or replace trigger t_io_vPessoas

instead of insert

on vPessoas

declare v_cod_instrutor number;

begin

select max(cod_instrutor)+1 into v_cod_instrutor from
instrutores;

if :new.tipo = 'a' then --aluno!

insert into alunos (nome_aluno) values (:new.nome);

else

insert into instrutores (cod_instrutor, nome_instrutor)
values (v_cod_instrutor, :new.nome);

end if;

end;

Triggers Instead of

-- testes: um aluno

```
insert into vPessoas (nome, tipo) values ('Aluno', 'a');
```

-- testes: um instrutor

```
insert into vPessoas (nome, tipo) values ('Instrutor', 'i');
```

-- Consultando:

```
select * from instrutores where nome_instrutor = 'Instrutor';
```

```
select * from alunos where nome_aluno = 'Aluno';
```

-- Limpeza:

```
drop view vPessoas;
```

```
drop trigger t_io_vPessoas;
```

Tipos de Triggers DML

ação	escopo	tempo
insert	For each row	before
insert	For each row	after
insert	statement	before
insert	statement	after
update	For each row	before
update	For each row	after
update	statement	before
update	statement	after
delete	For each row	before
delete	For each row	after
delete	statement	before
delete	statement	after

Variáveis de ambiente

- Quando um trigger afeta várias linhas, podemos, consultar ou modificar os valores antigos e novos dos campos da linha sendo alterada, através das variáveis de ambiente:

:old representa a linha corrente prévia

:new representa a linha após a alteração

Quando usar as variáveis...

triggers	:new	:old
before	Consultar/alterar	consultar
after	consultar	consultar
statement	Proibido usar	Proibido usar

Outras facilidades...

As funções abaixo informam qual operação disparou o trigger:

- Inserting à insert into
- Updating à update
- Deleting à delete

Quando usar as variáveis...

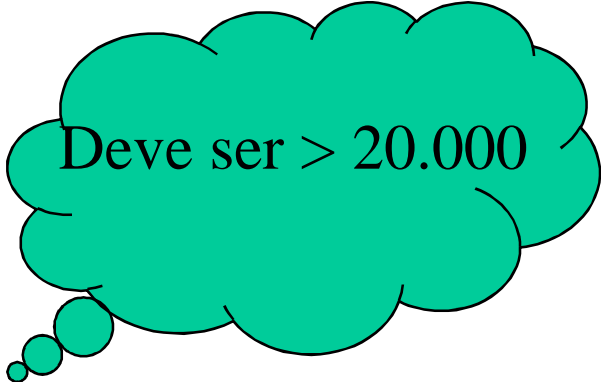
triggers	:new	:old
before	Consultar/alterar	consultar
after	consultar	consultar
statement	Proibido usar	Proibido usar

Como criar?

```
CREATE OR REPLACE TRIGGER nome_trigger  
[BEFORE | AFTER | INSTEAD OF]  
[INSERT OR DELETE OR UPDATE] [OF coluna]  
ON [ tabela | DATABASE | SCHEMA ]  
  WHEN condição  
[FOR EACH ROW]  
Bloco PL/SQL
```

Exemplo

```
create or replace trigger t_aft_upd_row_AumentaPrecos
after update
on cursos
for each row
begin
    if :new.preco > 1200 then
        raise_application_error(-20500, 'Tentativa
        exagerada de aumento!');
    end if;
end;
/
```



Deve ser > 20.000

Exemplo

Possível Execução:

```
SQL> update cursos set preco=preco*2;
```

```
update cursos set preco=preco*2
```

```
*
```

```
ERROR at line 1:
```

```
ORA-20500: Tentativa exagerada de aumento!
```

```
ORA-06512: at
```

```
"MARILDE.T_AFT_UPD_ROW_AUMENTAPRECOS",  
line 3
```

```
ORA-04088: error during execution of  
trigger
```

```
'MARILDE.T_AFT_UPD_ROW_AUMENTAPRECOS'
```

```
/
```

Dicas

- Para confirmar a criação do Trigger:

```
select trigger_name from user_triggers;
```

- Para eliminar:

```
drop trigger nome_trigger;
```

- Para desabilitar/habilitar:

```
alter table tabela disable |enable all triggers;
```

```
alter trigger nome_trigger disable |enable ;
```

Exemplo - Insert

Exemplo de disparo da trigger:

Insert into alunos (nome_aluno) values ('Chico Xavier');

- *Agora o trigger*

```
create or replace trigger t_bef_ins_row_InserereAluno
  before insert
  on alunos
  for each row
declare
  nova_matricula number;
begin
  select Gera_Matr_aluno.Nextval
  into nova_matricula from dual;

  :new.matricula := nova_matricula;
end;
/
```

Exer

```
create or replace  
t_bef_upd_stm  
before update  
on cursos
```

```
Begin
```

```
update Tab_Auditoria
```

```
set atualizacoes = atualizacoes + 1;
```

```
end;
```

```
/
```

Exemplo de disparo da trigger:

Update cursos set preco=100;

Comprovando:

Select * from tab_auditoria;

Obs.: a tabela tab_auditoria já deve existir!

Exe

```
create or replace  
  trigger t_bef_del_row  
  before delete  
  on turmas  
  for each row  
begin  
  delete historico  
  where cod_turma = :old.cod_turma;  
end;  
/
```

- Antes de disparar o trigger:
Alter table historico disable constraint historico_turma_fk;

- Exemplo de disparo da trigger:

Delete turmas;

- Comprovando:

*Select * from turmas;*

*Select * from historico;*

Obs.: dê um rollback para restabelecer os valores!

Exemplo – múltiplos eventos

```
create or replace trigger t_bef_updIns_stm_MultHist
  before insert or update
  on historico
declare
```

Exemplo de disparo da trigger:

```
be
Insert into historico values (12,12,10);--em horário impróprio
Update historico set nota = 10;
```

```
  if inserting then
    if v_agora > 1830 then
      raise_application_error(-20600, 'Hora proibida para
inserções');
    end if;
  else
    if v_hoje = 1 then
      raise_application_error(-20700, 'Dia proibido para
atualizações');
    end if;
  end if;
end; /
```


Trigger Solução

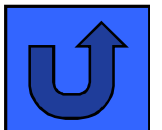
```
-- Limpeza:  
drop view vPessoas;  
drop trigger t_io_vPessoas;
```

```
values (ora_dict_obj_name);
```

```
-- testes: aluno  
insert into vPessoas (nome, tipo) values ('Aluno', 'a');  
  
-- testes: um instrutor  
insert into vPessoas (nome, tipo) values ('Instrutor', 'i');
```

-- Consultando:

```
select * from instrutores where nome_instrutor = 'Instrutor';  
select * from alunos where nome_aluno = 'Aluno';
```



Cursor

- Representa uma tabela temporariamente armazenada em memória e criada como resultado dos comandos: *select*, *insert*, *update*, *delete*, *commit* ou *rollback*.
- Contém os registros afetados pelo comando que provocou sua criação.
- ***Explícitos:*** gerados apenas pelo *Select*, deve ser declarado e manipulado via comandos próprios.
- **Implícitos:** dispensam qualquer tipo de tratamento.

Atributos de Cursores

Sql%rowcount	Informa quantas linhas foram afetadas pelo comando que gerou o cursor.
Sql%found	Será true caso alguma linha tenha sido afetada.
Sql%notfound	Será false caso alguma linha tenha sido afetada.
Sql%isopen	Será true caso o cursor esteja aberto (cursores explícitos).

Em cursores **explícitos** a palavra **SQL** é trocada pelo **nome do cursor**.

Exemplo de Cursor Implícito

```
create or replace function
  exclui_instrutores_cursor_imp
return varchar2
is
begin
  delete instrutores
    where cod_instrutor not in
      (select distinct cod_instrutor from
turmas);
  if sql%found then
return ('Foram eliminados: ' ||
to_char(sql%rowcount) || ' instrutores');
  else
return ('Nenhum instrutor eliminado.');
```

/

Visualizando o resultado...

Set serveroutput on; *//variável de ambiente.*

```
declare saida varchar2(40);  
begin  
    saida:=exclui_instrutores_cursor_imp;  
    dbms_output.put_line('Saida: '||saida);  
end;  
/
```

Comandos Cursor Explícito

Open	Cria fisicamente a tabela temporária e posiciona o ponteiro de leitura no primeiro registro.
Fetch	Carrega para variáveis locais o conteúdo da linha indicada pelo ponteiro de leitura.
Close	Fecha o cursor.

Exemplo Cursor Explícito

create or replace procedure Classifica_Cursos_Cur_Exp

IS

```
cursor ccursos is select nome_curso, preco from cursos;
```

```
v_nome_curso cursos.nome_curso%type;
```

```
v_preco      cursos.preco%type;
```

```
v_classifica varchar2(10);
```

BEGIN

```
open ccursos;
```

```
fetch ccursos into v_nome_curso, v_preco;
```

```
while ccursos%found loop
```

```
    if v_preco < 300 then v_classifica := 'Barato';
```

```
    elsif v_preco < 600 then v_classifica := 'Médio';
```

```
    else v_classifica := 'Caro';
```

```
end if;
```

```
    dbms_output.put_line ('Curso:  ' || v_nome_curso || ' é ' ||
```

```
v_classifica);
```

```
        fetch ccursos into v_nome_curso, v_preco;
```

```
end loop;
```

```
close ccursos;
```

END;

/

Cursor Parametrizado

```
create or replace procedure Class_Cursos_Cur_Exp_Param  
(v_valor_minimo number)
```

```
IS
```

```
    cursor ccursos (v_valor_minimo in number) is  
        select nome_curso, preco from cursos  
        where preco > v_valor_minimo;
```

```
    v_nome_curso cursos.nome_curso%type;
```

```
    v_preco cursos.preco%type;
```

```
    v_classificavvarchar2(10);
```

```
BEGIN
```

```
    open ccursos (v_valor_minimo);
```

```
    fetch ccursos into v_nome_curso, v_preco;
```

```
    while ccursos%found loop
```

```
        if v_preco < 300 then v_classifica := 'Barato';
```

```
        elsif v_preco < 600 then v_classifica := 'Médio';
```

```
        else v_classifica := 'Caro';
```

```
        end if;
```

```
        fetch ccursos into v_nome_curso, v_preco;
```

```
    end loop;
```

```
    close ccursos;
```

```
END;
```

```
/
```

