

Laboratório de Bases de Dados

PROF. JOSÉ FERNANDO RODRIGUES JR

AULA 4 – PL/SQL (PROCEDURAL LANGUAGE/STRUCTURED QUERY LANGUAGE)

MATERIAL ORIGINAL EDITADO: PROFA. ELAINE PARROS MACHADO DE SOUSA

Contexto de programação

1 GL – linguagem de máquina, 0's e 1's

2 GL – *assembly*, mnemônicos como LOAD e STORE

3 GL – de alto nível, como C, Java, ...

4 GL – declarações que abstraem os algoritmos e estruturas, como SQL

5 GL – programação baseada em requisitos

PL/SQL

PL/SQL combina as construções e procedimentos da linguagem Ada (3GL) com a flexibilidade do SQL (4 GL)

- estende SQL:
 - variáveis e tipos
 - estruturas de controle
 - procedimentos e funções
 - tipos de objeto e métodos

PL/SQL

PL/SQL
proced
flexibi

Outras combinações 3GL/4GL:

- PostgreSQL – **PL/pgSQL**
- IBM DB2 – **SQL PL**
- Microsoft SQL Server - **Transact-SQL**

- estende SQL
- variáveis
- estruturas de controle
- procedimentos e funções
- tipos de objeto e métodos

Princípios básicos PL/SQL

Estrutura em 3 blocos

DECLARE

*/*variáveis, tipos, cursores, subprogramas, ... */*

BEGIN

/ instruções... */*

EXCEPTION

*/*tratamento de exceções*/*

END;

Princípios básicos PL/SQL

Declaração/Inicialização de Variáveis

```
nome [CONSTANT] tipo [NOT NULL] [DEFAULT] [:= valor]
```

Princípios básicos PL/SQL

Exemplo

```
SET SERVEROUTPUT ON;  
DECLARE  
    v_count NUMBER;  
BEGIN  
    SELECT count(*) INTO v_count FROM Aluno;  
    dbms_output.put_line('Total = ' || v_count);  
END;
```

Status dos comandos SQL

SQL%

Utilizado para auditar as instruções:

- UPDATE
- SELECT ... INTO
- INSERT
- DELETE

embutidas no código PL/SQL

➔ O SQL% é chamado de cursor implícito

Exemplo – Cursor Implícito

DECLARE

v_nota CONSTANT NUMERIC(4,2) := 5.0;

BEGIN

UPDATE matricula SET nota = v_nota
WHERE nota > 3.0 AND nota < 6.0
AND sigla= 'SSC0722';

/* COMO SABER O RESULTADO DA INSTRUÇÃO? */
/* R.: pergunte ao SQL%*/

END;

Exemplo – Cursor Implícito

DECLARE

v_nota CONSTANT NUMERIC(4,2) := 5.0;

BEGIN

UPDATE matricula SET nota = v_nota
WHERE nota > 3.0 AND nota < 6.0
AND sigla = 'SC540';

/*cursor implícito SQL associado ao UPADATE recebe dados*/

IF SQL%FOUND --(alguma tupla foi atualizada?)

THEN dbms_output.put_line(SQL%ROWCOUNT || ' alunos
tiveram a nota alterada'); --(quantas tuplas?)

ELSE dbms_output.put_line('Nenhum aluno teve a nota
alterada');

END IF;

END; /*a consolidação dos dados depende de commit*/

Exemplo – Cursor Implícito

DECLARE

v_nota CONSTANT NUMERIC(4,2) := 5.0;

BEGIN

UPDATE

W

/*cursor

IF SQL%

THEN db

ELSE dbms_output.put_line('Nenhum aluno teve a nota
alterada');

END IF;

END; /*a consolidação dos dados depende de commit*/

SQL% guarda as informações da última instrução apenas – uma nova instrução apaga os dados da anterior.

dados*/

os
tuplas?)

SQL%

INSERT/UPDATE/DELETE

- **FOUND**
 - **TRUE**: se o comando anterior alterou alguma tupla
 - **FALSE**: caso contrário
- **NOTFOUND** (**! FOUND**)
- **ROWCOUNT**: nro de linhas alteradas pelo comando anterior
- **ISOPEN**
 - sempre **FALSE** – propriedade útil apenas para cursores explícitos

Variáveis e controle de fluxo

Tipagem automática de variáveis

DECLARE

```
v_nome ALUNO.NOME%TYPE;  
v_cpf PROFESSOR.NFUNC%TYPE;
```

Equivale a:

DECLARE

```
v_nome VARCHAR(100);  
v_cpf NUMERIC(10);
```

➔ O %TYPE faz com que o SGBD descubra qual é o tipo daquele dado no bd.

■ Exemplo – **SELECT INTO**

```
set serveroutput on;
```


```
DECLARE
```

```
    v_nome ALUNO.NOME%TYPE;
```

```
    v_nusp ALUNO.NUSP%TYPE;
```

```
BEGIN
```

```
    SELECT nome, nusp INTO v_nome, v_nusp
       FROM aluno A
      WHERE A.nusp = 1;
```



```
    dbms_output.put_line('Nome ' || v_nome || ', NUSP ' || v_nusp);
```

```
EXCEPTION /* exceções associadas ao SELECT INTO */
```

```
    WHEN NO_DATA_FOUND THEN
```

```
        dbms_output.put_line('Aluno não encontrado');
```

```
    /*se nusp não fosse único...*/
```

```
    WHEN TOO_MANY_ROWS THEN
```

```
        dbms_output.put_line('Há mais de um aluno
                               com este numero');
```

```
END;
```


■ Exemplo – **SELECT INTO**

LEMBRE-SE: o SELECT ... INTO só funciona com dados de EXATAMENTE uma única tupla.

-Se nenhuma tupla é retornada → **NO_DATA_FOUND**

-Se mais de uma tupla é retornada → **TOO_MANY_ROWS**

Tipagem automática de variáveis

DECLARE

v_professor PROFESSOR%ROWTYPE;

Equivale a:

DECLARE

```
TYPE professor_record_type IS RECORD (  
    NFunc          PROFESSOR.NFunc%TYPE,  
    Nome           PROFESSOR.Nome%TYPE,  
    Idade          PROFESSOR.Idade%TYPE,  
    Titulacao      PROFESSOR.Titulacao%TYPE  
);
```

v_professor professor_record_type;

➔ O %ROWTYPE faz com que o SGBD descubra qual é o tipo de tuplas inteiras, isto é, de todos os seus atributos


■ Exemplo – SELECT INTO

DECLARE

`v_professor Professor%ROWTYPE;`

BEGIN

`SELECT * INTO v_professor
FROM Professor A
WHERE A.nfunc = 10;`



`dbms_output.put_line('Nome ' || v_professor.nome ||
' , CPF ' || v_professor.nfunc);`

EXCEPTION /* exceções associadas ao SELECT INTO */

`WHEN NO_DATA_FOUND THEN`

`dbms_output.put_line('Professor não encontrado');`

`WHEN TOO_MANY_ROWS THEN`

`dbms_output.put_line('Há mais de um professor
com este NFunc');`

END;

Controle de fluxo PL/SQL

Controle de fluxo PL/SQL

- `IF ... THEN END IF;`
- `IF ... THEN ELSE ... END IF;`
- `IF ... THEN`
 `ELSIF ... THEN...`
 `ELSE ... END IF;`
- `CASE <variável>`
 `WHEN <valor> THEN <instruções>`
 `WHEN ... THEN...`
 `....`
 `ELSE ... /*opcional*/`
 `END CASE;`

■ Exemplo - INSERT

Exemplo:

Deseja-se matricular um aluno na turma 1 do ano atual da disciplina SSC0722, para tanto:

1) A turma deve existir

2) A turma não pode ter mais do que 5 alunos matriculados

■ Exemplo - INSERT

```
DECLARE
    v_count_turma NUMBER;
    v_count_aluno  NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count_turma FROM TURMA L
        WHERE L.Sigla = 'SM500';

    IF v_count_turma = 0 THEN
        INSERT INTO TURMA VALUES('SM500',1,0);
        dbms_output.put_line('Nova turma criada');
    END IF;

    SELECT COUNT(*) INTO v_count_aluno FROM matricula M
        WHERE M.sigla = 'SM500' and
            M.ano = EXTRACT (YEAR FROM SYSDATE) and M.Numero = 1;

    IF v_count_aluno < 5 THEN
        INSERT INTO matricula(Sigla,Numero,Aluno,Ano)
        VALUES ('SM500',1,1, EXTRACT (YEAR FROM SYSDATE));
        dbms_output.put_line('Aluno matriculado');

    ELSE dbms_output.put_line('Turma lotada');
    END IF;
END;
```

■ Exemplo - INSERT

```
DECLARE
```

```
  v_count_turma NUMBER;
```

```
  v_count_aluno  NUMBER;
```

```
BEGIN
```

```
  SELECT COUNT(*) INTO v_count_turma FROM turma T
    WHERE L.Sigla = 'SM500';
```

1) Total de turmas SC500 do ano atual

```
  IF v_count_turma = 0 THEN
```

```
    INSERT INTO TURMA VALUES ('SM500',1,0);
```

```
    dbms_output.put_line('Nova turma criada');
  END IF;
```

Se o total == 0, a turma não existe e deve ser criada.

```
  SELECT COUNT(*) INTO v_count_aluno FROM matricula M
```

```
    WHERE M.sigla = 'SM500' and
```

```
      M.ano = EXTRACT (YEAR FROM SYSDATE);
```

2) Total de alunos da turma - no máximo 5.

```
  IF v_count_aluno < 5 THEN
```

```
    INSERT INTO matricula(Sigla,Numero,Aluno,Ano)
```

```
    VALUES ('SM500',1,1, EXTRACT (YEAR FROM SYSDATE));
```

```
    dbms_output.put_line('Aluno matriculado');
  ELSE dbms_output.put_line('Turma lotada');
```

Se o total de alunos < 5, cabem mais alunos – matricula o novo aluno.

```
  END IF;
```

```
END;
```

```
END;
```


■ Exemplo – Tratamento de Exceção

```
DECLARE
```

```
    v_count_aluno NUMBER;
```

```
BEGIN
```

```
INSERT INTO matricula(Sigla,Numero,Aluno,Ano)
```

```
    VALUES ('SM600',1,1, EXTRACT (YEAR FROM SYSDATE));
```

```
/*Erro de integridade: não há turma SSC0722-2017*/
```

```
EXCEPTION
```

```
    WHEN OTHERS
```

```
        THEN dbms_output.put_line('Erro nro:  ' || SQLCODE  
                                   || '. Mensagem: ' || SQLERRM );
```

```
END;
```

Loops PL/SQL

Estruturas de Repetição

- `LOOP <instruções>`
 `EXIT WHEN <condição de parada>`
 `END LOOP;`
- `WHILE <condição de parada> LOOP`
 `<instruções>`
 `END LOOP;`
- `FOR <contador> IN [REVERSE] <min>..<max>`
 `LOOP <instruções>`
 `END LOOP;`

Exemplo

DECLARE

v disciplina TURMA.SIGLA%TYPE;

```
v anoTurma    TURMA.NUMERO%TYPE;
```

BEGIN

```
v_disciplina := 'SM500';
```

```
/* cria 6 turmas para a disciplina SSC0600*/
```

FOR nroTurma IN 1..6 LOOP

```
INSERT INTO TURMA VALUES (v disciplina,nroTurma,0);
```

```
dbms output.put line('Turma ' || nroTurma || ' criada.');
```

END LOOP;

EXCEPTION

WHEN OTHERS

```
THEN dbms_output.put_line('Erro nro: ' || SQLCODE
                           || '. Mensagem: ' || SQLERRM );
```

END ;

Cursor

Cursor

Passos:

1) declarar o cursor

2) abrir o cursor

- **OPEN**

3) buscar resultados

- **FETCH** – retorna uma tupla por vez e avança para a próxima no conjunto ativo

4) fechar cursor

- **CLOSE**

Exemplo – Cursor

```
set serveroutput on;
```

```
DECLARE
```

```
    CURSOR c_professores IS SELECT * FROM PROFESSOR;
```

```
    v_professor c_professores%ROWTYPE;
```

```
BEGIN
```

```
    OPEN c_professores;    /*abre cursor - executa consulta */
```

```
    LOOP
```



```
        FETCH c_professores INTO v_professor; /*recupera tupla*/
```

```
        /*sai do loop se não há mais tuplas*/
```

```
        EXIT WHEN c_professores%NOTFOUND;
```

```
        dbms_output.put_line('CPF: ' || v_professor.nfunc ||
```

```
                               ' - Nome: ' || v_professor.nome);
```

```
    END LOOP;
```

```
    CLOSE c_professores; /*fecha cursor*/
```

```
END;
```

Cursor - Sintaxe simplificada

```
set serveroutput on;
DECLARE
    CURSOR c_professores IS SELECT * FROM PROFESSOR;
BEGIN
    FOR v_professor IN c_professores LOOP
        dbms_output.put_line('CPF: ' || v_professor.NFUNC ||
                               ' - Nome: ' || v_professor.nome);
    END LOOP;
END;
```

Exemplo – Compondo operações dinamicamente

```
SET SERVEROUTPUT ON;
DECLARE
  CURSOR c_professores IS SELECT * FROM PROFESSOR;
BEGIN
  FOR v_professor IN c_professores LOOP
    dbms_output.put_line('NFunc: ' || v_professor.NFunc ||
                        ' - Nome: ' || v_professor.Nome);

    UPDATE PROFESSOR
    SET Idade = trunc(DBMS_RANDOM.value(25, 70))
    WHERE NFunc = v_professor.NFunc;

    dbms_output.put_line('Idade do professor NFunc ' ||
                        v_professor.NFunc || ' atualizada.');
```

END LOOP;

END;

Cursor

Atributos do tipo *CURSOR*

- **FOUND**
- **NULL** se ainda não houve nenhum **FETCH**
- **true** se o **FETCH** anterior retornou uma tupla
- **false** caso contrário
- **NOTFOUND: !FOUND**
- **ISOPEN**
- **ROWCOUNT**
 - nro de tuplas **já lidas** por **FETCH**

Cursor

DECLARE

CURSOR c_professores IS SELECT * FROM PROFESSOR;

v_total_processed INTEGER := 0;

v_professor PROFESSOR%ROWTYPE;

BEGIN

OPEN c_professores;

LOOP

FETCH c_professores INTO v_professor;

IF c_professores%**FOUND** THEN

v_total_processed := c_professores%**ROWCOUNT**;

dbms_output.put_line('NFunc: ' || v_professor.NFunc ||

' - Nome: ' || v_professor.Nome ||

' - Idade: ' || v_professor.Idade ||

' - Titulacao: ' || v_professor.Titulacao);

ELSE EXIT;

END IF;

END LOOP;

CLOSE c_professores;

dbms_output.put_line('Total de professores processados: ' || v_total_processed);

END;

Cursor for update

```
DECLARE
    CURSOR c_aluno IS
        SELECT NUSP, Cidade
        FROM Aluno
        WHERE Idade > 18
        FOR UPDATE OF Cidade;

    v_cidade Aluno.Cidade%TYPE;
BEGIN
    FOR aluno_rec IN c_aluno LOOP
        -- Atualiza a cidade para 'São Paulo' se não for 'São Paulo'
        IF aluno_rec.Cidade != 'São Paulo' THEN
            UPDATE Aluno
            SET Cidade = 'São Paulo'
            WHERE CURRENT OF c_aluno; -- não precisa recuperar a chave
        END IF;
    END LOOP;

    COMMIT;
END;
```

Procedimentos e Funções

- Subprogramas PL/SQL
 - armazenados no SGBD
 - locais
 - em código PL/SQL anônimo
 - em subprogramas armazenados

Procedimentos armazenados

```
CREATE OR REPLACE PROCEDURE AtualizaTitulacaoProfessor(  
    p_nfunc IN Professor.NFunc%TYPE,  
    p_nova_titulacao IN Professor.Titulacao%TYPE) IS  
BEGIN  
    -- Verifica se a titulação atual é diferente da nova titulação  
    IF EXISTS (SELECT 1 FROM Professor WHERE NFunc = p_nfunc  
                AND Titulacao != p_nova_titulacao) THEN  
        -- Atualiza a titulação do professor  
        UPDATE Professor  
        SET Titulacao = p_nova_titulacao  
        WHERE NFunc = p_nfunc;  
        COMMIT;  
  
        DBMS_OUTPUT.PUT_LINE('Titulação atualizada com sucesso.');    ELSE  
        DBMS_OUTPUT.PUT_LINE('Nenhuma atualização foi realizada.');    END IF;  
END AtualizaTitulacaoProfessor;  
  
-- -----  
BEGIN  
    AtualizaTitulacaoProfessor(12345, 'Doutorado');END;
```

Procedimentos armazenados

```
CREATE OR REPLACE PROCEDURE AtualizaTitulacaoProfessor(  
    p_nfunc IN Professor.NFunc%TYPE,  
    p_nova_titulacao IN Professor.Titulacao%TYPE) IS  
BEGIN  
    -- Informa que o parâmetro é só de entrada e não será alterado  
    -- Verifica se a titulação atual é diferente da nova titulação  
    IF EXISTS (SELECT 1 FROM Professor WHERE NFunc = p_nfunc  
        AND Titulacao != p_nova_titulacao) THEN  
        -- Atualiza a titulação do professor  
        UPDATE Professor  
        SET Titulacao = p_nova_titulacao  
        WHERE NFunc = p_nfunc;  
        COMMIT;  
  
        DBMS_OUTPUT.PUT_LINE('Titulação atualizada com sucesso.');    ELSE  
        DBMS_OUTPUT.PUT_LINE('Nenhuma atualização foi realizada.');    END IF;  
END AtualizaTitulacaoProfessor;  
  
-- -----  
BEGIN  
    AtualizaTitulacaoProfessor(12345, 'Doutorado');  
END;
```

Procedimentos armazenados

```
CREATE OR REPLACE PROCEDURE ContagemAlunosDisciplina(  
    p_sigla IN Disciplina.Sigla%TYPE,  
    p_num_alunos OUT NUMBER  
) IS  
BEGIN  
    -- Conta o número de alunos matriculados na disciplina  
    SELECT COUNT(DISTINCT Aluno) INTO p_num_alunos  
    FROM Matricula  
    WHERE Sigla = p_sigla;  
  
    -- output opcional  
    --DBMS_OUTPUT.PUT_LINE('Número de alunos:' || p_num_alunos);  
END ContagemAlunosDisciplina;
```

Informa que o parâmetro é só de saída e um valor não precisa ser passado, mas uma variável sim

```
-- -----  
DECLARE  
    v_num_alunos NUMBER;  
BEGIN  
    ContagemAlunosDisciplina('CS101', v_num_alunos);  
    DBMS_OUTPUT.PUT_LINE('Alunosmatriculados: ' || v_num_alunos);  
END;
```

Procedures e Functions

Parâmetros – Modos

- **IN (padrão)**
 - parâmetro: atua como uma constante
 - valor passado: constante, variável inicializada, literal ou expressão
- **OUT**
 - parâmetro: atua como uma variável não inicializada; **tem valor NULL dentro do proc/func, mesmo que um valor tenha sido passado de fora**
 - valor passado: variável, o que for feito dentro do proc/func é visto do lado de fora
- **IN OUT**
 - parâmetro: atua como uma variável inicializada, vê-se o que foi atribuído fora
 - valor passado: variável, o que for feito dentro do proc/func é visto do lado de fora

Funções armazenadas


```
CREATE OR REPLACE FUNCTION MediaNotasDisciplina(  
    p_sigla IN Disciplina.Sigla%TYPE  
) RETURN NUMBER IS  
    v_media NUMBER;  
BEGIN  
    SELECT AVG(Nota) INTO v_media  
    FROM Matricula  
    GROUP BY Sigla;  
  
    RETURN v_media;  
END MediaNotasDisciplina;  
-- -----  
DECLARE  
    v_media NUMBER;  
BEGIN  
    v_media := MediaNotasDisciplina('SM500');  
    DBMS_OUTPUT.PUT_LINE('Média de notas da disciplina: ' || v_media);  
END;  
ou  
SELECT MediaNotasDisciplina('SM500'); AS MEDIA  
FROM DUAL;
```

Funções armazenadas - Retornando múltiplos valores

CREATE OR REPLACE PACKAGE escola IS

 *Construção simples para organizar vários objetos lógicos*

TYPE top_alunos_t IS RECORD (--dentro de um package o tipo fica no dicionário de dados

primeiro VARCHAR2(100),  *Tipos RECORD, em particular, precisam estar dentro de pacotes para serem usados amplamente*
segundo VARCHAR2(100),
terceiro VARCHAR2(100)


);

FUNCTION melhores_alunos RETURN top_alunos_t;

END escola;

Funções armazenadas - Retornando múltiplos valores

CREATE OR REPLACE PACKAGE BODY escola IS

 *O corpo completa o pacote*

FUNCTION melhores_alunos RETURN top_alunos_t IS

alunos top_alunos_t;

BEGIN

alunos.primeiro := 'Carlos';

alunos.segundo := 'Joao';

alunos.terceiro := 'Maria';

RETURN alunos;

END melhores_alunos;

END escola;

DECLARE

alunos escola.top_alunos_t;

BEGIN

alunos := **escola.melhores_alunos()**;

-- Acessa os campos individualmente usando a notação de ponto

DBMS_OUTPUT.PUT_LINE(alunos.primeiro);

DBMS_OUTPUT.PUT_LINE(alunos.segundo);

DBMS_OUTPUT.PUT_LINE(alunos.terceiro);

END;

Cursor parametrizável

CURSOR EXPLÍCITO PARAMETRIZÁVEL

- Nos exemplos vistos até aqui, o cursor deve ser conhecido em tempo de compilação
- Pode ser útil parametrizar o cursor para se determinar os dados a serem consultados de acordo com a necessidade, em tempo de execução
- Deseja-se parametrizar o cursor

REF CURSORS

- Cursor usual:

```
CURSOR c_cursor_normal IS  
    SELECT COUNT (*)  
    FROM L01_MORADOR  
    WHERE MCPF > 3;
```

REF CURSORS

- Cursor usual em um procedure:

```
CREATE OR REPLACE PROCEDURE UsaCursor(pTotal OUT NUMBER) AS
    /*Retorna uma única tupla com um único valor, computada
       da mesma maneira em todas as execuções*/
    CURSOR c_cursor_normal IS
        SELECT COUNT(*)
        FROM L01_MORADOR
        WHERE MCPF > 3;

BEGIN
    OPEN c_cursor_normal;
    FETCH c_cursor_normal INTO pTotal;
    CLOSE c_cursor_normal;

END UsaCursor;
```

CURSOR EXPLÍCITO PARAMETRIZÁVEL

- Exemplo:

create or replace

PROCEDURE testeCursor(**N** NUMBER) AS

CURSOR c_cursor IS (SELECT * FROM ALUNO WHERE NUSP > **N**);

v_cursor c_cursor%ROWTYPE;

BEGIN

OPEN c_cursor;

LOOP

FETCH c_cursor INTO v_cursor; /*recupera tupla*/

EXIT WHEN c_cursor%NOTFOUND;

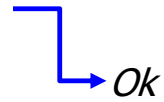
dbms_output.put_line(v_cursor.nome);

END LOOP;

close c_cursor;

END;

call testeCursor(1000000);



CURSOR EXPLÍCITO PARAMETRIZÁVEL

- Exemplo:

create or replace

PROCEDURE testeCursor(**N** NUMBER, **TABLENAME** VARCHAR2) AS

CURSOR c_cursor IS (SELECT * FROM **TABLENAME** WHERE NUSP > **N**);

v_cursor c_cursor%ROWTYPE;

BEGIN

OPEN c_cursor;

LOOP

FETCH c_cursor INTO v_cursor; /*recupera tupla*/

EXIT WHEN c_cursor%NOTFOUND;

dbms_output.put_line(v_cursor.nome);

END LOOP;

close c_cursor;

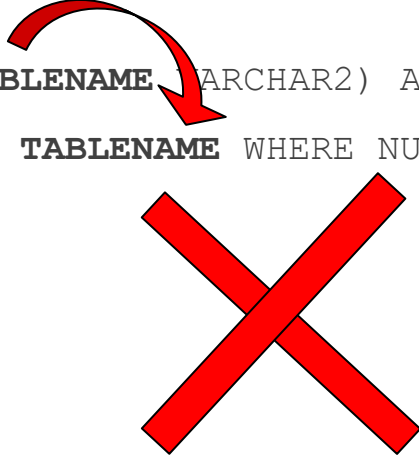
END;



CURSOR EXPLÍCITO PARAMETRIZÁVEL

- Exemplo:

```
create or replace  
  
PROCEDURE testeCursor(N NUMBER, TABLENAME VARCHAR2) AS  
  
  CURSOR c_cursor IS (SELECT * FROM TABLENAME WHERE NUSP = N);  
  
  v_cursor c_cursor%ROWTYPE;  
  
BEGIN  
  
  OPEN c_cursor;  
  
  LOOP  
  
    FETCH c_cursor INTO v_cursor; /*recupera tupla*/  
  
    EXIT WHEN c_cursor%NOTFOUND;  
  
    dbms_output.put_line(v_cursor.nome);  
  
  END LOOP;  
  
  close c_cursor;  
  
END;
```



REF CURSORS

Problema:

O cursor normal **não é totalmente parametrizável** – não se pode, por exemplo, escolher quais atributos, nem qual tabela.

REF CURSORS

- Solução: REF CURSORS

```
p_cursor SYS_REFCURSOR;
```

REF CURSORS

```
create or replace PROCEDURE consultar_tabela(  
    p_atributos IN VARCHAR2, p_tabela IN VARCHAR2,  
    p_atributo_filtro IN VARCHAR2, p_valor_filtro IN VARCHAR2) IS  
  
    v_cursor SYS_REFCURSOR;  
    v_sql VARCHAR2(1000);  
    v_resultado VARCHAR2(1000);  
  
BEGIN  
    v_sql := 'SELECT ' || p_atributos ||  
            ' FROM ' || p_tabela ||  
            ' WHERE ' || p_atributo_filtro || ' = ' || p_valor_filtro;  
    OPEN v_cursor FOR v_sql;  
    LOOP  
        FETCH v_cursor INTO v_resultado;  
        EXIT WHEN v_cursor%NOTFOUND;  
        DBMS_OUTPUT.PUT_LINE('Resultado: ' || v_resultado);  
    END LOOP;  
    CLOSE v_cursor;  
END consultar_tabela;
```

REF CURSORS

```
BEGIN
```

```
    consultar_tabela(  
        p_tabela => 'Aluno',  
        p_atributo_filtro => 'Cidade',  
        p_valor_filtro => 'São Paulo',  
        p_atributos_retornar => 'Nome'  
    );
```

```
END;
```

```
/
```

```
BEGIN
```

```
    consultar_tabela(  
        p_atributos => 'Nome',  
        p_tabela => 'Professor',  
        p_atributo_filtro => 'Titulacao',  
        p_valor_filtro => 'Mestrado'  
    );
```

```
END; /
```

EXECUTE IMMEDIATE

- Execução de **SQL definido dinamicamente**, isto é, SQL do qual não se sabe, em tempo de projeto, nomes de dados, colunas, e predicados
- Permite a execução de DML, DDL, blocos, e comandos em geral

EXECUTE IMMEDIATE

```
CREATE OR REPLACE PROCEDURE criar_e_preencher_tabela(
```

```
  p_tabela_nome IN VARCHAR2,
```

```
  p_colunas IN VARCHAR2) IS
```

```
BEGIN
```

```
-- Construir a consulta SQL para criar a tabela
```

```
EXECUTE IMMEDIATE 'CREATE TABLE ' || p_tabela_nome || ' (' || p_colunas || ')';
```

```
DBMS_OUTPUT.PUT_LINE('Tabela ' || p_tabela_nome || ' criada com sucesso.');
```

```
-- Inserir dados dinamicamente na nova tabela
```

```
EXECUTE IMMEDIATE 'INSERT INTO ' || p_tabela_nome || ' VALUES ("Alice", 23)';
```

```
EXECUTE IMMEDIATE 'INSERT INTO ' || p_tabela_nome || ' VALUES ("Bob", 30)';
```

```
EXECUTE IMMEDIATE 'INSERT INTO ' || p_tabela_nome || ' VALUES ("Carol", 27)';
```

 *Aceita o envio de declarações SQL quaisquer para execução imediata*

```
DBMS_OUTPUT.PUT_LINE('Dados inseridos na tabela ' || p_tabela_nome || '.');
```

```
COMMIT;
```

```
END criar_e_preencher_tabela;
```

Prática 4

A solid orange horizontal bar spanning the width of the slide at the bottom.