



# Laboratório de Bases de Dados

Prof. José Fernando Rodrigues Júnior

## **Aula 6 – PL/SQL (Procedural Language/Structured Query Language)**

Material original editado: Profa. Elaine Parros Machado de Sousa



# Contexto de programação

---

- **1GL** – 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 (constraints)



# 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 Outras combinações 3GL/4GL:

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

■ es

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

s e  
(3GL)



# 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*]

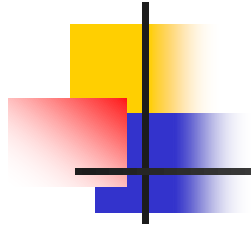


# Princípios básicos PL/SQL

---

## ■ Exemplo

```
SET SERVEROUTPUT ON;  
DECLARE  
    v_count NUMBER;  
BEGIN  
    SELECT count(*) INTO v_count FROM L01_MORADOR;  
    dbms_output.put_line('NMoradores = ' || v_count);  
END;
```



# Cursor Implícito





# Cursor Implícito - SQL

---

- Utilizado para auditar as instruções:
  - UPDATE
  - SELECT . . . INTO
  - INSERT
  - DELETE

embutidas no código PL/SQL



# Cursor Implícito - SQL

---

- Todas as instruções SQL são executadas dentro de uma área de contexto, então...
  - existe um **cursor implícito** que aponta para essa área de contexto → **cursor SQL**
- PL/SQL implicitamente abre o cursor SQL, processa a instrução SQL e fecha o cursor

# Exemplo – Cursor Implícito

DECLARE

v\_nota CONSTANT lbd08\_matricula.nota%TYPE := 5.0;

BEGIN

UPDATE lbd08\_matricula SET nota = v\_nota  
WHERE nota > 3.0 AND nota < 6.0  
AND coddisc = 'SSC0722';

/\* COMO SABER O RESULTADO DA INSTRUÇÃO? \*/

/\* R.: pergunte ao cursor implícito SQL \*/

END;

# Exemplo – Cursor Implícito

DECLARE

v\_nota CONSTANT lbd08\_matricula.nota%TYPE := 5.0;

BEGIN

UPDATE lbd08\_matricula SET nota = v\_nota  
WHERE nota > 3.0 AND nota < 6.0  
AND coddisc = 'SSC0722';

/\*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;

# Exemplo – Cursor Implícito

DECLARE

v\_nota CONSTANT lbd08\_matricula.nota%TYPE := 5.0;

BEGIN

UPDATE

W

/\*cursor

IF SQL%

THEN db

ELSE dbms\_output.put\_line(

O cursor implícito SQL guarda as informações da última instrução realizada – uma nova instrução apaga os dados da anterior.

dados\*/

os  
tuplas?)

Nenhum aluno teve a nota alterada');

END IF;

END;



# Cursor Implícito - 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



# Cursor Implícito - SQL

---

## ■ SELECT INTO

### ■ FOUND

- **TRUE**: se o comando anterior retornou alguma tupla
- **FALSE**: caso contrário – no entanto a **exceção NO\_DATA\_FOUND** é lançada imediatamente

### ■ NOTFOUND

- **!FOUND**

### ■ ROWCOUNT: nro de tuplas retornadas

- se `#tuplas = 0` → `ROWCOUNT == 0` **exceção NO\_DATA\_FOUND** - acessível apenas no bloco de exceção
- se `#tuplas > 1` **exceção TOO\_MANY\_ROWS** - acessível apenas no bloco de exceção com `ROWCOUNT = 1`
- se `#tuplas = 1` → ok, `ROWCOUNT = 1`

### ■ ISOPEN

- sempre `FALSE` – propriedade útil apenas para cursores explícitos



---

# **Variáveis e controle de fluxo**



## ■ Exemplo



---

```
DECLARE
```

```
    v_nome L01_MORADOR.MNOME%TYPE;
```

```
    v_cpf  L01_MORADOR.MCPF%TYPE;
```

Equivale a:

```
DECLARE
```

```
    v_nome VARCHAR2(100);
```

```
    v_cpf  NUMBER(11,0);
```

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

## ■ Exemplo – **SELECT INTO**

```
set serveroutput on;
```

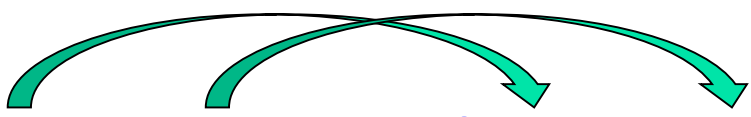
```
DECLARE
```

```
    v_nome L01_Morador.mnome%TYPE;
```

```
    v_cpf L01_Morador.mcpf%TYPE;
```

```
BEGIN
```

```
    SELECT mnome, mcpf INTO v_nome, v_cpf
    FROM L01_Morador A
    WHERE A.mcpf = 1;
```



```
    dbms_output.put_line('Nome ' || v_nome ||
                          ', CPF ' || v_cpf);
```

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

```
    WHEN NO_DATA_FOUND THEN
```

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

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

```
    WHEN TOO_MANY_ROWS THEN
```

```
        dbms_output.put_line('Há mais de um morador
                               com este CPF');
```

```
END;
```

## ■ Exemplo – **SELECT INTO**

```
set serveroutput on;
```

```
DECLARE
```

```
    v_nome L01_Morador.mnome%TYPE;
```

```
BEGIN
```

```
SELECT
```

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**

```
dbms
```

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

```
    WHEN NO_DATA_FOUND THEN
```

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

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

```
    WHEN TOO_MANY_ROWS THEN
```

```
        dbms_output.put_line('Há mais de um morador  
                                com este CPF');
```

```
END;
```

## ■ Exemplo

```
DECLARE  
    v_vinculo L01_MORADOR%ROWTYPE;
```

Equivale a:

```
DECLARE  
    v_cpf NUMBER(11,0), v_tipo, VARCHAR2(12), v_nome  
    VARCHAR2(110,0), ...
```

➔ 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_morador L01_Morador%ROWTYPE;`

BEGIN

`SELECT * INTO v_morador  
FROM L01_Morador A  
WHERE A.mcpf = 1;`



`dbms_output.put_line('Nome ' || v_morador.mnome ||  
' , CPF ' || v_morador.mcpf);`

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

`WHEN NO_DATA_FOUND THEN`

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

`/*se cpf não fosse único...*/`

`WHEN TOO_MANY_ROWS THEN`

`dbms_output.put_line('Há mais de um morador  
com este CPF');`

END;



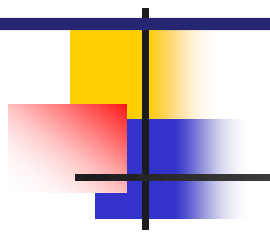
# Princípios básicos PL/SQL

---

- Estruturas de controle de fluxo

- `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 lbd07_TURMA L
        WHERE L.CODDISC = 'SSC0722' and
            L.ano = EXTRACT (YEAR FROM SYSDATE) and L.NROTURMA = 1;

    IF v_count_turma = 0 THEN
        INSERT INTO LBD07_TURMA VALUES(1,EXTRACT (YEAR FROM SYSDATE), 'SSC0722',31);
        dbms_output.put_line('Nova turma criada');
    END IF;

    SELECT COUNT(*) INTO v_count_aluno FROM lbd08_matricula M
        WHERE M.CODDISC = 'SSC0722' and
            M.ano = EXTRACT (YEAR FROM SYSDATE) and M.NROTURMA = 1;

    IF v_count_aluno < 5 THEN
        INSERT INTO lbd08_matricula(NROUSP,CODDISC,ANO,NROTURMA,NOTA)
        VALUES (1,'SSC0722',EXTRACT (YEAR FROM SYSDATE),1, 0);
        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 lbd07_turma T
    WHERE L.CODDISC = 'SSC0722' and
          L.ano = EXTRACT (YEAR FROM SYSDATE);
```

1) Total de turmas SSC0722 do ano atual, da turma 1 (deve ser igual a 1)

```
  IF v_count_turma = 0 THEN
```

```
    INSERT INTO lbd07_TURMA VALUES (1, EXTRACT (YEAR FROM SYSDATE));
    dbms_output.put_line('Nova turma criada');
```

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

```
  END IF;
```

```
  SELECT COUNT(*) INTO v_count_aluno FROM lbd08_matricula M
    WHERE M.CODDISC = 'SSC0722' and
          M.ano = EXTRACT (YEAR FROM SYSDATE) and M.NROTURMA = 1;
```

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

```
  IF v_count_aluno < 5 THEN
```

```
    INSERT INTO lbd08_matricula (NROU, CODDISC, ANO, NROTURMA)
    VALUES (1, 'SSC0722', EXTRACT (YEAR FROM SYSDATE), 1);
    dbms_output.put_line('Aluno matriculado');
```

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

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

```
  END IF;
```

```
END;
```

# ■ Exemplo – Tratamento de Exceção

DECLARE

    v\_count\_aluno NUMBER;

BEGIN

INSERT INTO lbd08\_matricula(NROUSP,CODDISC,ANO,NROTURMA,NOTA)

    VALUES (6,'SSC0722',EXTRACT (YEAR FROM SYSDATE),1, 0);

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

EXCEPTION

    WHEN OTHERS

        THEN dbms\_output.put\_line('Erro nro:   ' || SQLCODE  
                                  || '. Mensagem: ' || SQLERRM );

END;

# ■ Exemplo - Exceção

DECLARE

    v\_count\_aluno NUMBER;

    exc\_lotada EXCEPTION;

BEGIN

    SELECT COUNT(\*) INTO v\_count\_aluno FROM lbd08\_matricula M  
        WHERE M.CODDISC = 'SSC0722' and  
              M.ano = 2009 and M.NROTURMA = 1;

    IF v\_count\_aluno < 5 THEN

        INSERT INTO lbd08\_matricula(NROUSP,CODDISC,ANO,NROTURMA,NOTA)

        VALUES (6,'SSC0722',2009,1, 0);

        ELSE RAISE exc\_lotada;

    END IF;

EXCEPTION

    WHEN exc\_lotada

        THEN dbms\_output.put\_line('Erro nro: ' || SQLCODE

                                  || '. Mensagem: ' || SQLERRM

                  || ' Exceção: turma lotada!');

    WHEN OTHERS

        THEN dbms\_output.put\_line('Erro nro: ' || SQLCODE

                                  || '. Mensagem: ' || SQLERRM );

END;



# Princípios básicos 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 LBD07\_TURMA.CODDISC%TYPE;

    v\_anoTurma    LBD07\_TURMA.ANO%TYPE;

BEGIN

    v\_disciplina := 'SSC0723';

    v\_anoTurma := EXTRACT (YEAR FROM SYSDATE);

    /\* cria 6 turmas para a disciplina SSC0722\*/

    FOR nroTurma IN 1..6 LOOP

        INSERT INTO LBD07\_TURMA

            VALUES (nroTurma, v\_anoTurma, v\_disciplina, 31);

        dbms\_output.put\_line('Turma ' || nroTurma || ' criada.');

    END LOOP;

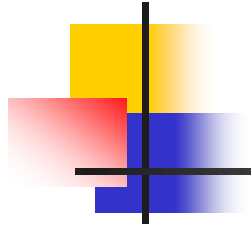
EXCEPTION

    WHEN OTHERS

        THEN dbms\_output.put\_line('Erro nro: ' || SQLCODE

                                  || '. Mensagem: ' || SQLERRM );

END;



# Cursor Explícito



# Cursores

---

- **Área de contexto**

- área de memória com informações de processamento de uma instrução
- inclui **conjunto ativo**  $\Rightarrow$  linhas retornadas por uma consulta

- **Cursor**

- *handle* para uma área de contexto (cursor **NÃO** é uma variável de memória)
- tipos:
  - Implícito  $\rightarrow$  já visto
  - explícito



# Cursor Explícito

---

- 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 Explícito

```
set serveroutput on;
```

```
DECLARE
```

```
    CURSOR c_moradores IS SELECT * FROM L01_MORADOR;
```

```
    v_moradores c_moradores%ROWTYPE;
```

```
BEGIN
```

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

```
    LOOP
```

```
        FETCH c_moradores INTO v_moradores; /*recupera tupla*/
```

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

```
        EXIT WHEN c_moradores%NOTFOUND;
```

```
        dbms_output.put_line('CPF: ' || v_moradores.mcpf ||  
                               ' - Nome: ' || v_moradores.mnome);
```

```
    END LOOP;
```

```
    CLOSE c_moradores; /*fecha cursor*/
```

```
END;
```

# Exemplo – Cursor Explícito

## Sintaxe simplificada

```
set serveroutput on;
DECLARE
    CURSOR c_moradores IS SELECT * FROM L01_MORADOR;
BEGIN
    FOR v_moradores IN c_moradores LOOP
        dbms_output.put_line('CPF: ' || v_moradores.mcpf ||
                               ' - Nome: ' || v_moradores.mnome);
    END LOOP;
END;
```

# Exemplo – Compondo operações dinamicamente

```
set serveroutput on;
DECLARE
  CURSOR c_moradores IS SELECT * FROM L01_MORADOR;
BEGIN
  FOR v_moradores IN c_moradores LOOP
    dbms_output.put_line('CPF: ' || v_moradores.mcpf ||
                        ' - Nome: ' || v_moradores.mnome);

    UPDATE L01_MORADOR
    SET nrofamiliares = trunc(DBMS_RANDOM.value(1,10))
    WHERE MCPF = v_moradores.mcpf;
    dbms_output.put_line('Numero de familiares do CPF ' ||
                        v_moradores.mcpf || ' atualizado.');
```

END LOOP;

END;



# Cursor Explícito

---

- 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**



# Procedimentos e Funções

---

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

# Procedimentos armazenados

```
create or replace
PROCEDURE insere_matricula (
    p_turma D17_Matricula.TURMA%TYPE,
    p_aluno D17_Matricula.Aluno%TYPE,  /*pode ser IS*/
    p_disciplina D17_Matricula.DISCIPLINA%TYPE,
    p_ano D17_Matricula.ano%TYPE,
    p_semestre D17_Matricula.semestre%TYPE) AS

    v_count NUMBER;
    e_lotada EXCEPTION;
BEGIN
    SELECT COUNT(*) INTO v_count FROM D17_matricula M
        WHERE M.DISCIPLINA = p_disciplina AND M.TURMA = p_turma AND
            M.ano = EXTRACT (YEAR FROM SYSDATE) AND M.semestre = 1;
    IF v_count < 70 THEN /*verifica se a turma não está lotada*/
        INSERT INTO D17_matricula VALUES (p_turma,p_aluno,p_disciplina, 0.0,
            0, 'M',p_ano, 1); /* -> insere*/
    ELSE RAISE e_lotada; /*lotada -> exceção*/
    END IF;
EXCEPTION
    WHEN e_lotada
        THEN raise_application_error(-20001, 'Turma lotada');
    WHEN OTHERS
        THEN raise_application_error(-20001, 'Erro detectado: ' || SQLCODE
            || '-' || SQLERRM);
END insere_matricula;
```

# Procedimentos armazenados

---

```
/*Programa Principal - PL/SQL anônimo*/
```

```
BEGIN
```

```
    Insere_matricula (2,6022222,'scc1212',2012,1);
```

```
END;
```

OU

```
call Insere_matricula (2,6022222,'scc1212',2012,1);
```

# Funções armazenadas

```
CREATE OR REPLACE FUNCTION media (  
    p_aluno D17_MATRICULA.Aluno%TYPE )  
    RETURN NUMBER IS /*pode ser AS*/  
  
    v_media NUMBER;  
  
BEGIN  
    SELECT AVG(nota) INTO v_media FROM D17_MATRICULA  
        WHERE aluno = p_aluno;  
  
    RETURN v_media; /* RETURN obrigatório para sair da função*/  
  
END media;
```

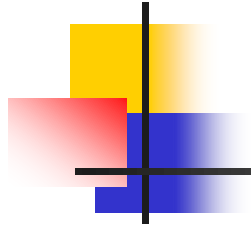


# Funções armazenadas

```
SELECT MEDIA(5888888) AS MEDIA  
FROM DUAL
```

OU

```
/*Programa Principal - PL/SQL anônimo*/  
DECLARE  
    v_media NUMBER;  
    v_aluno D17_Matricula.Aluno%TYPE;  
BEGIN  
    v_aluno := 5888888;  
    v_media := media(v_aluno);  
  
    dbms_output.put_line('Média de ' || v_aluno || ' = '  
        || v_media);  
END;
```



# Cursor Explícito parametrizável



# CURSOR EXPLÍCITO PARAMETRIZÁVEL

---

- Nos exemplos vistos até aqui, o cursor deve ser conhecido em tempo de compilação
- No entanto, pode ser útil parametrizar o cursor para se determinar os dados a serem consultados de acordo com a necessidade
- Deseja-se parametrizar o cursor



# REF CURSORS

---

- Cursor normal:

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



# REF CURSORS

---

- Cursor normal 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 D01_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(1);
```



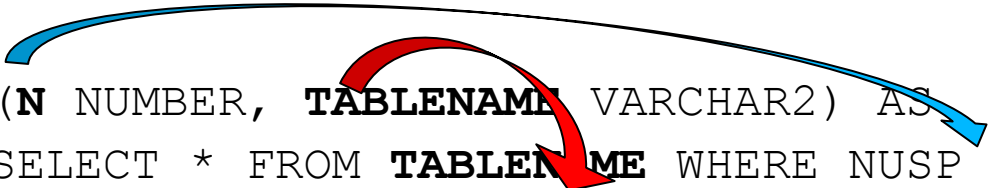


# CURSOR EXPLÍCITO PARAMETRIZÁVEL

---

- 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;
```

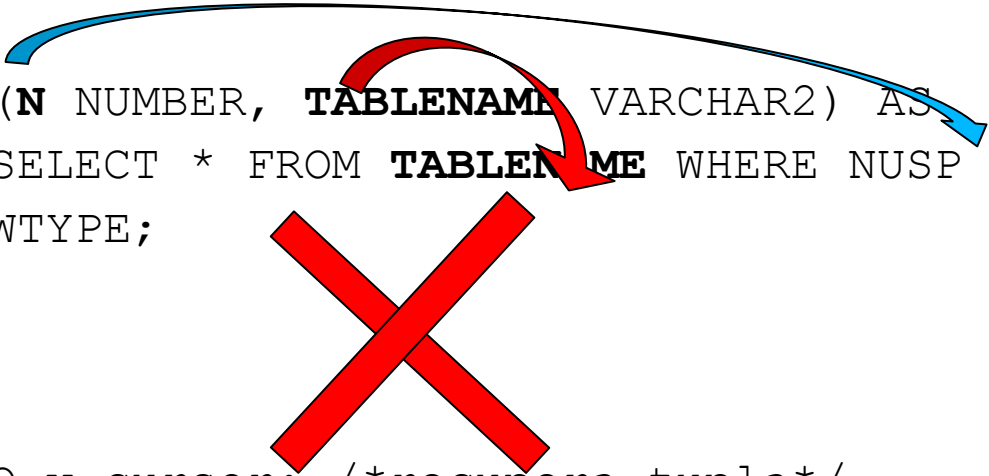




# CURSOR EXPLÍCITO PARAMETRIZÁVEL

- 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;
```





## Problemas:

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

```
CREATE OR REPLACE
PROCEDURE testeCursor(N NUMBER, a_table VARCHAR2) AS
  CURSOR c_cursor IS (SELECT * FROM a_table WHERE NUSP = N);
  v_cursor c_cursor%ROWTYPE;
BEGIN
  FOR v_cursor IN c_cursor LOOP
    dbms_output.put_line(v_cursor.nome);
  END LOOP;
END; → não compila
```

2) Se eu quiser usar (retornar) um **conjunto**, com mais de uma tupla, **do lado de fora do procedimento**, eu não consigo

```
FETCH c_cursor_normal INTO pTotal;
CLOSE c_cursor_normal;
```

```
END UsaCursor;
```



# REF CURSORS

---

## ■ Solução: REF CURSORS

```
TYPE TIPO_REF_CURSOR IS REF CURSOR;
```

Define-se seu tipo ou usa-se **SYS\_REFCURSOR** para o mesmo propósito

```
PROCEDURE UsaCursor(pTotal OUT NUMBER,  
                    p_cursor_de_saida OUT SYS_REFCURSOR,  
                    p_valor_CPF IN NUMBER) ;
```

# REF CURSORS

CREATE OR REPLACE

PROCEDURE **UsaCursor**(

p\_cursor\_de\_saida **OUT** **SYS\_REFCURSOR**,

p\_table\_name VARCHAR2,

p\_nome\_atributo VARCHAR2,

p\_valor IN NUMBER) AS

sql\_text VARCHAR2(200);

cTemp D04\_PROFESSOR%ROWTYPE;

BEGIN

**sql\_text** := 'SELECT \* FROM ' ||

p\_table\_name || ' WHERE ' ||

p\_nome\_atributo || ' > ' || p\_valor;

DBMS\_OUTPUT.PUT\_LINE(**sql\_text**);

**OPEN** p\_cursor\_de\_saida FOR **sql\_text**;

LOOP

FETCH p\_cursor\_de\_saida INTO cTemp;

EXIT WHEN p\_cursor\_de\_saida %NOTFOUND;

dbms\_output.put\_line(cTemp.NFUN || ' - ' || cTemp.Nome);

END LOOP;

CLOSE p\_cursor\_de\_saida;

END UsaCursor;

# REF CURSORS

declare

```
p_cursor SYS_REFCURSOR;
p_cursor_interno SYS_REFCURSOR;
p_table_name VARCHAR2(100) := 'F01_ESTADO';
p_nome_atributo VARCHAR2(100) := 'SIGLA';
sql_text VARCHAR2(200);
sql_text_interno VARCHAR2(200);
cTemp F01_ESTADO%ROWTYPE;
cTemp_interno F02_cidade%rowtype;
BEGIN
    sql_text := 'SELECT * FROM ' || p_table_name;
    DBMS_OUTPUT.PUT_LINE(sql_text);
    OPEN p_cursor FOR sql_text;
    LOOP
        FETCH p_cursor INTO cTemp;
        EXIT WHEN p_cursor%NOTFOUND;
        dbms_output.put_line('-----');
        dbms_output.put_line(cTemp.sigla || ' - ' || cTemp.estado);
    END LOOP;
```

# REF CURSORS

```
    sql_text_interno := 'SELECT * FROM F02_CIDADE WHERE SIGLAC =''' ||  
cTemp.sigla||''';  
    open p_cursor_interno for sql_text_interno;  
    fetch p_cursor_interno into cTemp_interno;  
    dbms_output.put_line(cTemp_interno.cidade || ' - ' ||  
cTemp_interno.siglac);  
    EXIT WHEN p_cursor_interno%NOTFOUND;  
    close p_cursor_interno;  
END LOOP;  
    CLOSE p_cursor;  
END UsaCursor;
```



# Prática 6