

# SISTEMAS DE BANCO DE DADOS 2

## AULA 13

PL / SQL

(Linguagem Procedural)

**ORACLE**

Vandor Roberto Vilardi Rissoli

Material original: Profa. Elaine P. M. de Sousa



# APRESENTAÇÃO

- *Programação Procedural em Banco de Dados Relacional*
- *Programando em PL/SQL*
- *Cursors*
  - *Explícito*
  - *Implícito*
- *Referências*



# Programação em Banco de Dados

A chave estrangeira é sempre definida na relação filho e a relação contendo o atributo referenciado (ou atributos) é a relação pai.

Para a definição desta restrição as palavras reservadas **FOREIGN KEY**, **REFERENCES** e **ON DELETE CASCADE** são empregadas, por exemplo:

```
CREATE TABLE CIDADE (  
    idCidade number          NOT NULL,  
    cidade varchar2(40)      NOT NULL,  
    estado varchar2(2),  
    CONSTRAINT CIDADE_PK PRIMARY KEY (idCidade),  
    CONSTRAINT CIDADE_ESTADO_FK FOREIGN KEY (estado)  
    REFERENCES ESTADO (sigla) ON DELETE CASCADE);
```

Identifica o(s) atributo(s)  
na relação filho

Identifica a relação pai  
e seu(s) atributo(s)

Identifica que as tuplas da relação filho  
também serão apagadas quando a tupla da  
Relação pai for apagada

# Programação em Banco de Dados

As opções de restrição relacionadas as tabelas relacionadas por uma chave estrangeira no **MySQL** variam conforme a necessidade da implementação, sendo possíveis:

- CASCADE: Atualiza ou exclui os registros da tabela filha automaticamente, ao atualizar ou excluir uma tupla da pai;
- RESTRICT: Rejeita a atualização ou exclusão de um registro da tabela pai, se houver registros na tabela filha;
- SET NULL: Define como NULL o valor do campo na tabela filha, ao atualizar ou excluir o registro da tabela pai;
- NO ACTION: Equivale à opção RESTRICT, mas a verificação de integridade referencial é executada após a tentativa de alterar a tabela. É a opção PADRÃO se nenhuma opção for definida na criação de chave estrangeira;
- SET DEFAULT: Define um valor padrão para coluna da tabela filha, aplicado quando um registro da tabela pai for atualizado ou excluído.

# Programação em Banco de Dados

Como já estudando, a restrição **CHECK** especifica uma condição que deverá ser satisfeita **para cada tupla da relação**, estando no nível de atributo ou da relação.

```
CREATE TABLE EMPREGADO (  
    matricula number          NOT NULL,  
    nome varchar2(50)         NOT NULL,  
    salario number(5,2)       NOT NULL,  
    CONSTRAINT EMPREGADO_PK PRIMARY KEY (matricula),  
    CONSTRAINT SALARIO_CK CHECK (salario > 1000 ) );
```

Um único atributo pode possuir várias restrições **CHECK**, não havendo limite no número destas restrições que podem ser definidas em um atributo (**nível só de tupla**).



# Programação em Banco de Dados

A restrição de **CHECK** pode usar as mesmas construções condicionais elaboradas para consultas (**SELECT**), existindo algumas exceções como:

- Não são permitidas referências às pseudocolunas CURRVAL, NEXTVAL, LEVEL e ROWNUM;

identificador inteiro do usuário corrente ↙

- Nem as chamadas as funções SYSDATE, UID, USER e USERENV; — retorna o nome do usuário corrente no BD — retorna informações de uma sessão corrente

- Nem as consultas que se referem a outros valores em outras tuplas... mas para isso existem alternativas, entre elas trabalhar com

**ASSERTION**



# Programação em Banco de Dados

## ASSERTION (SQL)

A implementação de regras de integridade **mais complexas** que trabalhem com **outras tuplas e resultados de consultas** (SELECT) não podem ser feitas pela restrição **CHECK**, mas por **ASSERTION**.

- Restrição que **NÃO** está amarrada a uma única tabela como as *constraints* de **CHECK**;
- Permite especificar restrições **mais complexas**
  - Envolve a comparação entre tuplas diferentes;
  - Analise comparativa com resultado de consultas (*select*);
- **Sempre** que alguma das **tabelas for alterada** a asserção é verificada pelo SGBD;
- Ações que **violam** a asserção **são rejeitadas** pelo SGBD.

# Programação em Banco de Dados

Suponha a restrição de integridade que NÃO permitirá o armazenamento de mais que um único empregado da empresa com o cargo de presidente:

```
CREATE ASSERTION um_unico_presidente AS CHECK  
( (SELECT COUNT(*)  
    FROM EMPREGADO e  
    WHERE e.cargo = 'presidente') <= 1  
);
```

Essa instrução SQL cria uma asserção que exige que não exista mais do que um único presidente entre os empregados da empresa.





# Programação em Banco de Dados

Suponha a restrição de integridade que permita o armazenamento somente do EMPREGADO que ganhe salário menor que seu SUPERVISOR.

```
CREATE ASSERTION salario_menor_supervisor AS CHECK  
(NOT EXISTS (SELECT *  
              FROM EMPREGADO e, EMPREGADO s  
              WHERE e.supervisor = s.matricula  
                    AND e.salario > s.salario)  
);
```

- Especifica uma consulta na asserção que recuperará os empregados que ganham mais que seu supervisor;
- A cláusula NOT EXISTS assegurará que nenhuma tupla seja recuperada;
- Se a consulta **NÃO** retornar **vazio** a restrição foi **violada**.

# Programação em Banco de Dados

Suponha agora a restrição de integridade de que nenhum EMPREGADO possa trabalhar em mais que 3 projetos.

```
CREATE ASSERTION limiteProjetos AS CHECK(  
  (NOT EXISTS ( SELECT t.matricula  
                FROM trabalha t  
                GROUP BY t.matricula  
                HAVING COUNT(*) > 3  
              )  
);
```

- Nenhum empregado conseguirá participar na empresa de mais que 3 projetos em suas atividades de trabalho;
- Se tal situação de mais que 3 projetos for tentado registrar o SGBD **rejeitará** tal armazenamento.



# Programação em Banco de Dados

- Instrução **DDL** (CREATE e DROP);
  - As restrições implementadas no SGBD por asserções promovem a **sobrecarga** (*overhead*) no SGBD;
    - Principalmente quando muitos usuários podem atualizar a base de dados do SGBD;
  - Vários SGBD **NÃO** oferecem suporte para implementação das **Asserções SQL**:
    - MySQL;
    - PostgreSQL;
    - SQLServer;
    - ORACLE e outros...
- Na versão 10g do **ORACLE** existe algo muito rudimentar, enquanto nas versões 11g (*grid*) e 12c (*cloud*) não oferecem suporte para asserções, mas os usuários tem sido inqueridos sobre este recurso que poderá estar na nova versão 13c.

# Programação em Banco de Dados

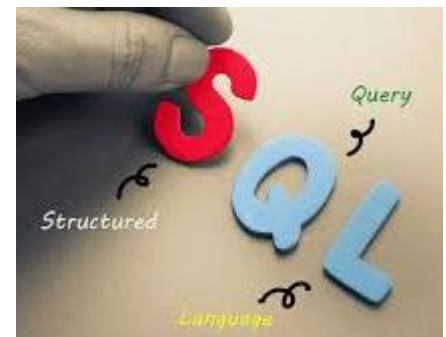
Veja a asserção criada para proibir que fornecedores das cidades de **Curitiba** ou **Blumenau** forneçam, em quantidades maiores que 50, as peças que custem R\$75,00 ou mais.

```
CREATE ASSERTION controleFornecedor AS CHECK  
( NOT EXISTS ( SELECT '1 fornecedor de todas as peças'  
                FROM FORNECEDOR f  
                WHERE f.cidade IN ('Curitiba', 'Blumenau')  
                AND NOT EXISTS (  
                    SELECT '1 produto não enviado'  
                    FROM PRODUTO p  
                    WHERE (p.preco >= 75)  
                    AND NOT EXISTS (  
                        SELECT '1 conexão'  
                        FROM ENTREGA e  
                        WHERE e.quantidade > 50  
                        AND e.idFornecedor = f.idFornecedor  
                        AND e.idProduto = p.idProduto) ) ) );
```



# Contexto de Programação

- **1GL** - Linguagem de máquina (binário em 0 e 1)
- **2GL** - Assembly, mnemônicos como LOAD e STORE
- **3GL** - Programação de alto nível, como C, Java, ...
- **4GL** - Declarações que abstraem os algoritmos e estruturas, como SQL
- **5GL** - Programação visual



# PL/SQL

- PL/SQL combina flexibilidade da SQL (4GL) com construções procedimentais do PL/SQL (3GL)
  - estende SQL:
    - variáveis e tipos
    - estruturas de controle
    - procedimentos e funções
    - tipos de objeto e métodos



# PL/SQL

- **PL/SQL *engine***  $\Rightarrow$  tecnologia
  - compila e executa blocos PL/SQL
  - pode ser instalado em:
    - **servidor Oracle**
      - *stored procedures e triggers*
      - blocos anônimos. Ex:
        - » **Ferramentas de desenvolvimento PL/SQL:** SQLPlus, SQL Developer, Rapid SQL, DBPartner, SQL Navigator, TOAD, SQL-Programmer, PL/SQL Developer, ...
        - » Pré-compiladores (ex: Pro\*C/C++), ODBC, JDBC, OCI
  - **ferramentas Oracle**
    - Oracle Forms
    - Oracle Reports

## **Outras combinações 3GL/4GL:**

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

# OPERADOR PL/SQL (*engine*)

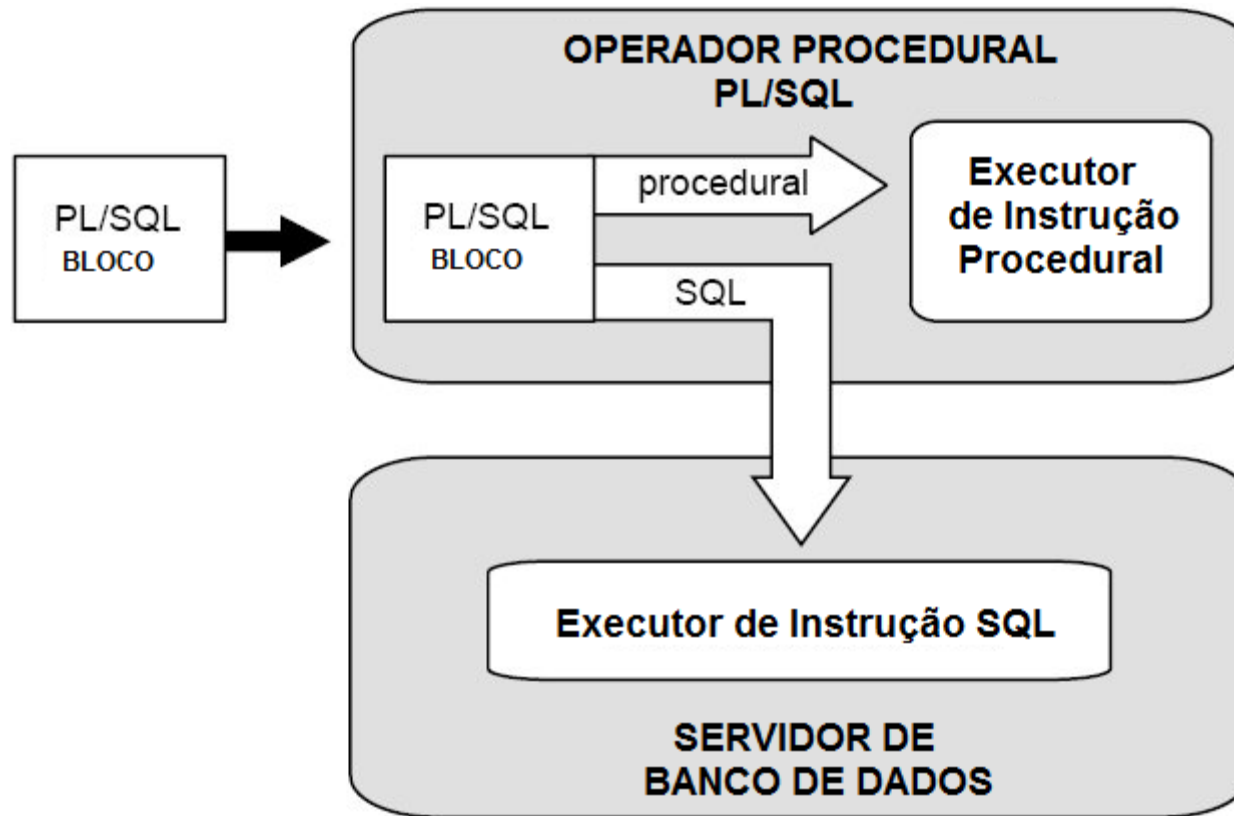


Figura baseada no *PL/SQL User's Guide and Reference (Release 2 (9.2))*

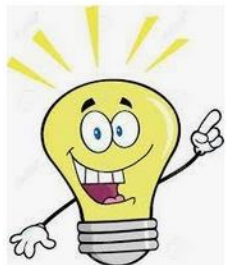




# PL/SQL

- **VANTAGENS**

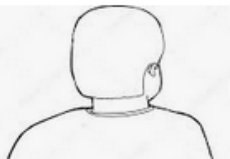
- suporte a SQL
- suporta a programação OO
- performance
- produtividade
- integração com **ORACLE**
- resolve “encruzilhadas” SQL
- definição de regras de negócio não abrangidas pelo projeto relacional



# PL/SQL

- **RECURSOS**

- estrutura em blocos
  - variáveis e tipos
  - tratamento de erros
  - estruturas de controle
    - condicionais
    - repetição
  - cursores
- procedimentos e funções
  - pacotes
  - coleções
  - conceitos OO



# 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*]

- Exemplo

```
SET SERVEROUTPUT ON;
```

```
DECLARE
```

```
    v_count NUMBER;
```

```
BEGIN
```

```
    SELECT count(*) INTO v_count FROM ALUNOS;
```

```
    dbms_output.put_line('Qtd.Alunos =' || v_count);
```

```
END;
```



# Princípios básicos PL/SQL

- Exemplo

**DECLARE**

`v_nome SBD2_VINCULO_UNB.nome%TYPE;`

`v_matricula SBD2_VINCULO_UNB.matricula%TYPE;`

**Equivale a:**

**DECLARE**

`v_nome VARCHAR2(100);`

`v_matricula NUMBER(9,0);`

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



- Exemplo - com **SELECT INTO**

```
set serveroutput on;
```

```
DECLARE
```

```
    v_nome MORADOR.mnome%TYPE;
```

```
    v_cpf   MORADOR.mcpf%TYPE;
```

```
BEGIN
```

```
    SELECT m.mnome, m.mcpf INTO v_nome, v_cpf
    FROM MORADOR m
    WHERE m.mcpf = 33125421039;
```

```
    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 - com **SELECT INTO**

```
set serveroutput on;
```

```
DECLARE
```

```
    v_nome MORADOR.mnome%TYPE;
```

```
    v_cpf  MORADOR.mcpf%TYPE;
```

```
BEGIN
```

```
    SELECT m.mnome, m.mcpf INTO v_nome, v_cpf
    FROM MORADOR m
    WHERE m.mcpf = 33125421039;
```

```
    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 - com **SELECT INTO**

```
set serveroutput on;
```

```
DECLARE
```

```
    v_nome MORADOR.mnome%TYPE;
```

```
    v_cpf   MORADOR.mcpf%TYPE;
```

```
BEGIN
```

```
    SELECT m.mnome, m.mcpf INTO v_nome, v_cpf
    FROM MORADOR m
    WHERE m.mcpf = 33125421039;
```

```
    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;
```





# Princípios básicos PL/SQL

- Exemplo

DECLARE

`v_vinculo SBD2_VINCULO_UNB%ROWTYPE;`

Equivale a:

DECLARE

`v_vinculo VARCHAR2(100),  
v_matricula NUMBER(9,0), ...`

➔ **%ROWTYPE** faz com que o SGBD descubra qual é o tipo de tuplas inteiras



- Exemplo - com **SELECT INTO**

```
DECLARE
```

```
    v_morador MORADOR%ROWTYPE;
```

```
BEGIN
```

```
    SELECT * INTO v_morador
           FROM MORADOR m
           WHERE m.mcpf = 33125421039;
```

```
    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;
```




- Exemplo - com **SELECT INTO**

DECLARE

**v\_morador MORADOR%ROWTYPE;**

BEGIN

SELECT \* **INTO** v\_morador  
FROM MORADOR m  
WHERE m.mcpf = 33125421039;



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 (condicional)

- 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;



# Princípios básicos PL/SQL

- Exemplo

Deseja-se matricular um aluno na turma 1 do ano atual da disciplina **FGA0060**, para tanto:

1) A turma deve existir

2) A turma não pode ter mais do que 50 alunos matriculados



# • Exemplo - de INSERT

DECLARE

    v\_count\_turma NUMBER;

    v\_count\_aluno NUMBER;

BEGIN

    SELECT COUNT(\*) INTO v\_count\_turma FROM SBD2\_TURMA t  
        WHERE t.codDisc = 'FGA0060' and  
              t.ano = EXTRACT (YEAR FROM SYSDATE) and t.nroTurma = 1;

    IF v\_count\_turma = 0 THEN

        INSERT INTO SBD2\_TURMA VALUES (1,EXTRACT(YEAR FROM SYSDATE), 'FGA0060',31);  
        dbms\_output.put\_line('Nova turma criada');

    END IF;

    SELECT COUNT(\*) INTO v\_count\_aluno FROM SBD2\_MATRICULA m  
        WHERE m.codDisc = 'FGA0060' and  
              m.ano = EXTRACT (YEAR FROM SYSDATE) and m.nroTurma = 1;

    IF v\_count\_aluno < 50 THEN

        INSERT INTO SBD2\_MATRICULA(matricula,codDisc,ano,nroTurma,nota)  
        VALUES (1,'FGA0060',EXTRACT (YEAR FROM SYSDATE),1, 0);  
        dbms\_output.put\_line('Aluno matriculado');

    ELSE dbms\_output.put\_line('Turma lotada');

    END IF;

END;

# • Exemplo - de INSERT

DECLARE

v\_count\_turma NUMBER;

v\_count\_aluno NUMBER;

BEGIN

```
SELECT COUNT(*) INTO v_count_turma FROM SBD2_TURMA t
WHERE t.codDisc = 'FGA0060' and
      t.ano = EXTRACT (YEAR FROM SYSDATE) and t.nroTurma = 1;
```

1) Total de turmas FGA0060 do ano atual, da turma 1 (deve ser igual a 1)

IF v\_count\_turma = 0 THEN

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

```
INSERT INTO SBD2_TURMA VALUES (1,EXTRACT(YEAR FROM SYSDATE), 'FGA0060',31);
dbms_output.put_line('Nova turma criada');
```

END IF;

2) Total de alunos da turma (no máximo 50).

```
SELECT COUNT(*) INTO v_count_aluno FROM SBD2_MATRICULA m
WHERE m.codDisc = 'FGA0060' and
      m.ano = EXTRACT (YEAR FROM SYSDATE) and m.nroTurma = 1;
```

IF v\_count\_aluno < 50 THEN

Se o total de alunos < 50, cabem mais alunos – matricula o novo aluno.

```
INSERT INTO SBD2_MATRICULA(matricula,codDisc,ano,nroTurma,nota)
VALUES (1,'FGA0060',EXTRACT (YEAR FROM SYSDATE),1, 0);
dbms_output.put_line('Aluno matriculado');
```

ELSE dbms\_output.put\_line('Turma lotada');

END IF;

END;

# • Exemplo - com EXCEÇÃO

DECLARE

    v\_count\_aluno NUMBER;

    exc\_lotada EXCEPTION;

BEGIN

    SELECT COUNT(\*) INTO v\_count\_aluno FROM SBD2\_MATRICULA m

        WHERE m.codDisc = 'FGA0060' and

            m.ano = EXTRACT (YEAR FROM SYSDATE) and m.nroTurma = 1;

    IF v\_count\_aluno < 50 THEN

        INSERT INTO SBD2\_MATRICULA(nroUnb,codDisc,ano,nroTurmaA,nota)

        VALUES (6,'FGA0060',EXTRACT (YEAR FROM SYSDATE),1, 0);

    ELSE RAISE exc\_lotada;

END IF;

EXCEPTION

    WHEN exc\_lotada

        THEN dbms\_output.put\_line('Turma lotada');

    WHEN OTHERS

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

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

END;



# • Exemplo - com EXCEÇÃO

DECLARE

`v_count_aluno` NUMBER;

`exc_lotada` EXCEPTION;

Total de alunos da turma (no máximo 50).

BEGIN

SELECT COUNT(\*) INTO `v_count_aluno` FROM SBD2\_MATRICULA m  
WHERE m.codDisc = 'FGA0060' and  
m.ano = EXTRACT (YEAR FROM SYSDATE) and m.nroTurma = 1;

Se o total de alunos < 50, cabem mais alunos –  
matricula o novo aluno.

IF `v_count_aluno` < 50 THEN  
INSERT INTO SBD2\_MATRICULA(nroUnb,codDisc,ano,nroTurmaA,nota)  
VALUES (6,'FGA0060',EXTRACT (YEAR FROM SYSDATE),1, 0);

ELSE RAISE `exc_lotada`;

END IF;

EXCEPTION

WHEN `exc_lotada`

THEN dbms\_output.put\_line('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 SBD2\_TURMA.codDisc%TYPE;

v\_anoTurma SBD2\_TURMA.ano%TYPE;

BEGIN

v\_disciplina := 'FGA0060';

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

/\* cria 6 turmas para a disciplina SCC103 \*/

FOR nroTurma IN 1..6 LOOP

INSERT INTO SBD2\_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;

# 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
  - explícito



# Cursor Explícito



**DECLARE**

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

**Result Set**

7369	SMITH	CLERK
7566	JONES	MANAGER
<b>7788</b>	<b>SCOTT</b>	<b>ANALYST</b>
7876	ADAMS	CLERK
7902	FORD	ANALYST

cursor → Current Row

Figura retirada de *PL/SQL User's Guide and Reference (Release 2 (9.2))*



# 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

DECLARE

```
CURSOR c_alunos IS SELECT * FROM SBD2_ALUNO;  
v_alunos c_alunos%ROWTYPE;
```

BEGIN

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

LOOP

```
    FETCH c_alunos INTO v_alunos;      /* recupera tupla */  
                                         /* sai do loop se não há mais tuplas */  
    EXIT WHEN c_alunos%NOTFOUND;
```

```
    dbms_output.put_line('Matricula:' || v_alunos.matricula  
                          || ' - Idade: ' || v_alunos.idade);
```

```
END LOOP;
```

```
CLOSE c_alunos;          /* fecha cursor - libera memória */
```

```
END;
```

/\* Exemplo de cursos EXPLÍCITO... para UPDATE \*/

DECLARE

```
CURSOR c_alunos IS SELECT m.nrousp, a.nome, m.nota
FROM SBD2_MATRICULA m JOIN SBD2_VINCULO_UNB a
ON m.nrousp = a.nrousp
WHERE m.codDisc= 'FGA0060' AND m.ano=2009 FOR UPDATE OF m.nota;
```

```
/* FOR UPDATE OF - registros ficam bloqueados para a seção corrente */
v_resultado c_alunos%ROWTYPE;          /* ROWTYPE associado a cursor */
```

BEGIN

OPEN c\_alunos;

LOOP

FETCH c\_alunos INTO v\_resultado;

EXIT WHEN c\_alunos%NOTFOUND;

```
dbms_output.put_line('Aluno: ' || v_resultado.nrousp || ' - ' ||
v_resultado.nome || ' Nota: ' || v_resultado.nota);
```

IF v\_resultado.nota = 4.99 THEN

UPDATE SBD2\_MATRICULA

SET nota = 5.0

WHERE CURRENT OF c\_alunos; /\* para update ou delete \*/

/\* CURRENT OF se refere necessariamente a um único registro \*/

/\* o uso é vinculado a cursores FOR UPDATE OF para update e delete \*/

END IF;

END LOOP;

COMMIT; /\* Release FOR UPDATE records \*/

CLOSE c\_alunos;

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**



# Exemplo - Cursor IMPLÍCITO

```
DECLARE
```

```
  v_nota CONSTANT SBD2_MATRICULA.nota%TYPE := 5.0;
```

```
BEGIN
```

```
  UPDATE SBD2_MATRICULA SET nota = v_nota  
    WHERE nota > 3.0 and nota < 6.0  
      AND codDisc = 'FGA0060';
```

```
  IF SQL%FOUND /* cursor implícito associado ao UPDATE */  
  THEN dbms_output.put_line(SQL%ROWCOUNT || ' alunos  
                                tiveram a nota alterada');  
  ELSE dbms_output.put_line('Nenhum aluno teve a nota  
                                alterada');  
  END IF;
```

```
END;
```



# 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



# Cursor Implícito - SQL

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



# 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 pelo comando anterior
  - 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



**Conclusão:** o **Oracle** só permite o uso de um **cursor** de seleção implícito caso ele selecione exatamente **uma única tupla**.

# Referência de Criação e Apoio ao Estudo

## Material para Consulta e Apoio ao Conteúdo

- Manual de consulta.
  - PL/SQL – User's Guide and Reference.
- Oracle Database11G SQL: Domine SQL e PL/SQL no banco de dados Oracle.
  - Livro
- François Oliveira -Apostila do Minicurso de PL/SQL
  - <https://docplayer.com.br/449445-Apostila-do-minicurso-de-pl-sql-francois-oliveira.html>

