FAESA
CENTRO UNIVERSITÁRIO

**Unidade de Engenharia e Computação**
Disciplina Banco de Dados – Profº. Howard Roatti

# Procedimentos Armazenados


Procedimentos Armazenados
Procedures
Triggers

# PROCEDURES

- São procedimentos armazenados no SGBD
- É um objeto no SGBD
- Podem receber parâmetros
- Podem chamar outras Procedures
- Podem retornar valores para os objetos chamadores
- Podem ser chamadas remotamente
- São objetos compilados e armazenados

# PROCEDURES

**CREATE** [OR REPLACE] **PROCEDURE** *procedure_name*
*[[( ]argument1 [mode] datatype*
*[, argument2 [mode] datatype...[ ) ]]*
{IS | AS}
*Bloco PL/SQL;*

# PROCEDURES

*procedure_name* → refere-se ao nome da procedure
*argument1* → refere-se ao nome de um parâmetro passado para a procedure
*mode* → define o tipo do parâmetro. Pode ser:
- IN (default) → passa o valor para a procedure
- OUT → recebe um valor vindo da procedure
- IN OUT → para um valor para a procedure e recebe dela um valor, possivelmente, diferente do que foi passado

*datatype* → refere-se ao tipo de dado do parâmetro
*Bloco PL/SQL* → refere-se ao corpo procedural que descreve as ações executadas pela procedure;

# PROCEDURES

Dicas sobre a sintaxe da Procedure:

1. Especifica a opção **REPLACE** quando a procedure já existir;
2. Use AS ou IS. São equivalente;
3. Inicie o bloco PL/SQL com a declaração da palavra **BEGIN** ou com a declaração das variáveis locais;
4. Nunca inicie o bloco PL/SQL de uma Procedure com a palavra **DECLARE**;
5. Termine o bloco PL/SQL com a palavra **END** seguida ou não pelo nome da procedure;
6. Argumentos e variáveis podem ser declaradas utilizando-se do **%TYPE** ou do **%ROWTYPE**

# PROCEDURES

Declarando Variáveis e Constantes:

**CREATE** [OR REPLACE] **PROCEDURE** *procedure_name*

```
v_nome                  varchar2(40);
v_telefone              number(8);
v_ativo                 boolean;
v_salario               number(8,2);
c_dias                  CONSTANT number(3) := 30;

BEGIN
  null;
END;
```

# PROCEDURES

## Estruturas de Controle

**IF-THEN**

```
Ex.: IF salario < 350 THEN
         salario := 350;
     END IF;
```

# PROCEDURES

Estruturas de Controle

**IF-THEN-ELSIF**

```
Ex.: IF salario < 350 THEN
         salario := 350;
     ELSIF salario > 350 and < 500 THEN
         salario := salario * 1.3;
     ELSE
         salario := salario * 1.2;
     END IF;
```

# PROCEDURES

Estruturas de Controle

**CASE-WHEN**

Ex.: CASE avaliacao

```
        WHEN 'A' THEN dbms_output.put_line('excelente');
        WHEN 'B' THEN dbms_output.put_line('bom');
        WHEN 'C' THEN dbms_output.put_line('regular');
        WHEN 'D' THEN dbms_output.put_line('insuficiente');
        ELSE bms_output.put_line('avaliação ausente');
    END CASE;
```

# PROCEDURES

Estruturas de Controle
**GOTO**

Ex.: GOTO desvio

```
<<desvio>>
    bms_output.put_line('desvio do programa');
```

# PROCEDURES

```
CREATE OR REPLACE PROCEDURE proc1
AS
BEGIN
        INSERT INTO PRO
        VALUES (1000, ‘kg’, ‘produto teste’, 1000.00, 1000);
END;
```

Objetivo da PROCEDURE?

# PROCEDURES

```
CREATE OR REPLACE PROCEDURE proc2
(a_codigo        in        integer,
 a_ue            in        varchar2,
 a_nome          in        varchar2,
 a_preco         in        numeric,
 a_quantidade    in        numeric)
AS
BEGIN
            INSERT INTO PRO
           VALUES (a_codigo, a_ue, a_nome, a_preco, a_quantidade);
END;
```

Objetivo da PROCEDURE?

# PROCEDURES

```
CREATE OR REPLACE PROCEDURE proc3
(a_codigo        in        pro.pro_cod%TYPE,
 a_nome          in        pro.pro_nome%TYPE,
 a_preco         in        pro.pro_preco%TYPE,
 a_quantidade    in        pro.pro_qt%TYPE)
AS
v_ue                       nvarchar2;
BEGIN
        SELECT ue_cod  INTO v_ue
            FROM ue WHERE upper(ue_desc) = 'LITRO';

         INSERT INTO PRO
        VALUES (a_codigo, v_ue, a_nome, a_preco, a_quantidade);
END;
```

# PROCEDURES

Controle de Transação:

1. Garante a integridade do Banco de Dados
2. Baseia-se nas quatro propriedades: ACID
3. Podem possuir declaração explicita ou implicita
4. Palavras chaves utilizadas: **BEGIN TRANSACTION (SQL Server)**, **COMMIT** e **ROLLBACK**

# PROCEDURES

```
CREATE OR REPLACE PROCEDURE proc4
AS
BEGIN
  INSERT INTO PRO
  VALUES (100, ‘kg’, ‘produto 100’, 100, 100 );

   COMMIT;

 EXCEPTION
    WHEN OTHERS THEN
            rollback;
            dbms_output.put_line (‘proc4’ || ‘erro no insert da tabela PRO’);
END;
```

# PROCEDURES

Funções PL/SQL - Definição

**SYSDATE** → obtem a data e hora do sistema

**USER** → obtem o usuário conectado

**ROWNUM** → limita o numero de linhas a serem retornada

**OPERADORES ARITMÉTICOS** → +, -, *, /

**FUNÇÕES NUMÉRICAS** →

ROUND → ARREDONDAMENTO DADOS NUMERICOS

TRUNK → TRUNCA DADOS NUMERICOS

MOD → RETORNA O RESTO DE UMA DIVISÃO

VE SE O NUMERO É PAR OU IMPAR

**FUNÇÕES DE CARACTER** →

UPPER → RETORNA O DADO EM CAIXA ALTA

LOWER → RETORNA O DADO EM CAIXA BAIXA

INITCAP → CAIXA ALTA NA PRIMEIRA LETRA

# PROCEDURES

Funções PL/SQL - Definição

**FUNÇÕES DE CARACTER →**

RTRIM → ELIMINA ESPACOS A DIREITA
LTRIM → ELIMINA ESPACOS A ESQUERDA
TRIM → ELIMINA ESPAÇOS A ESQUEDA E A DIREITA
RPAD → ADICIONA ESPACOS A UMA COLUNA
SUBSTR → CAPTURA PARTE DO CONTEUDO
LENGTH → CONTA A QUANTIDADE DE CARACTERES
REPLACE → SUBSTITUI CARACTER

**FUNÇÕES DE DATA →**

MONTHS_BETWEEN → DIFERENÇA ENTRE DATAS
EXTRACT → EXTRAI PARTE DE UMA DATA

# PROCEDURES

Funções PL/SQL - Definição

**FUNÇÕES DE DATA →** continuação

ADD_MONTHS → INCREMENTA DATA

LAST_DAY → MOSTRA O ULTIMO DIA DE UM MES

SYSTIMESTAMP → DATA DO SYSTEMA

**FUNÇÕES DE CONVERSÃO →**

TO_CHAR → CONVERTE PARA CARACTER

TO_NUMBER → CONVERTE PARA NUMERO

TO_DATE → CONVERTE PARA DATA

TO_TIMESTAMP → CONVERTE PARA TIMESTAMP

# PROCEDURES

Funções PL/SQL - exemplos

**SYSDATE** → SELECT **SYSDATE** FROM DUAL;

**USER** → SELECT **USER** FROM DUAL;

**ROWNUM** → SELECT * FROM PED WHERE **ROWNUM** < 10;

**OPERADORES ARITMÉTICOS** →

SELECT PRO_COD, PRO_PRECO ***** 1,5 FROM PRO;

# PROCEDURES

Funções PL/SQL - exemplos

**FUNÇÕES NUMÉRICAS →**

```
SELECT PRO_COD, ROUND (PRO_PRECO/30, 2)
FROM PRO

SELECT PRO_COD, TRUNK (PRO_PRECO/30, 2)
FROM PRO

SELECT PRO_COD, MOD (PRO_PRECO, 2)
FROM PRO
```

# PROCEDURES

Funções PL/SQL - exemplos

**FUNÇÕES DE CARACTER →**

SELECT **UPPER** (PRO_NOME), **LOWER**(UE_COD) FROM PRO;
SELECT **INITCAP** (PRO_NOME) FROM PRO;
SELECT **RTRIM** (PRO_NOME) FROM PRO;
SELECT **LTRIM** (PRO_NOME) FROM PRO;
SELECT **TRIM** (CLI_NOME) FROM CLI;
SELECT CLI_NOME, **RPAD** (CLI_FONE) FROM CLI;
SELECT **LENGTH** (PRO_NOME) FROM CLI;
SELECT **SUBSTR** (PRO_RUA, 4, 10) FROM CLI;
SELECT **REPLACE** (PRO_RUA, ‘RUA’, ‘LOGR’) FROM CLI;

# PROCEDURES

Funções PL/SQL - exemplos

**FUNÇÕES DE DATA →**

```
SELECT MONTHS_BETWEEN (PED_DT,PED_DT_EMB) "DIF“,
            EXTRACT (YEAR FROM PED_DT) “ANO”,
            EXTRACT (MONTH FROM PED_DT) “MES”,
            ADD_MONTHS (PED_DT, 3) “MES + 3”,
            LAST_DAY (PED_DT) “ULTIMO DIA”,
 FROM PED;

SELECT SYSTIMESTAMP FROM DUAL;
```

# PROCEDURES

Funções PL/SQL - exemplos

**FUNÇÕES DE CONVERSÃO →**

```
SELECT PED_COD, TO_CHAR(PED_DT, 'DL') FROM PED;

SELECT TO_CHAR(EXTRACT(YEAR FROM PED_DT)) ||
       TO_CHAR(EXTRACT(MONTH FROM PED_DT),'FM09') ||
       TO_CHAR(EXTRACT(DAY FROM PED_DT),'FM09')
                    "DtPedido"
 FROM PED;

SELECT TO_NUMBER('12345') + 300 FROM DUAL;
```

# PROCEDURES

Funções PL/SQL - exemplos

**FUNÇÕES DE CONVERSÃO →**

```
SELECT TO_DATE(’25-JAN-2007’, ‘DD-MON-RR’) FROM DUAL;
SELECT TO_DATE(’25-01-2007’, ‘DD-MM-RR’) FROM DUAL;

SELECT TO_TIMESTAMP('10-Sep-05 14:10:10.123000',
                                    'DD-Mon-RR HH24:MI:SS.FF')
        FROM DUAL;
```

```
CREATE OR REPLACE PROCEDURE proc5
(a_codigo          in          pro.pro_cod%TYPE,
 a_nome            in          pro.pro_nome%TYPE,
 a_preco           in          pro.pro_preco%TYPE,
 a_quantidade      in          pro.pro_qt%TYPE)
AS
v_ue                                 nvarchar2(3);
v_retorno     nvarchar2(1);
e_erro_proc6                         exception;
BEGIN
            SELECT ue_cod  INTO v_ue
                FROM ue WHERE upper(ue_desc) = 'LITRO';

            PROC6 (a_codigo, v_ue, a_nome, a_preco, a_quantidade, v_retorno);

            IF v_retorno = 1 THEN
                        raise e_erro_proc6;
            END IF;
            COMMIT;
EXCEPTION
    WHEN E_ERRO_PROC6 THEN
            rollback;      dbms_output.put_line ('erro no processamento da proc6');
    WHEN OTHERS THEN
            rollback;      dbms_output.put_line ('erro na proc5');

END;
```

# PROCEDURES

```
CREATE OR REPLACE PROCEDURE proc6
(a_codigo          in        pro.pro_cod%TYPE,
 a_ue              in        pro.ue_cod%TYPE,
 a_nome            in        pro.pro_nome%TYPE,
 a_preco           in        pro.pro_preco%TYPE,
 a_quantidade      in        pro.pro_qt%TYPE,
 a_retorno         out       nvarchar)
AS
BEGIN
  INSERT INTO PRO
  VALUES (a_codigo, a_ue, a_nome, a_preco, a_quantidade);

  EXCEPTION
    WHEN OTHERS THEN
            a_retorno := '1';
END;
```

# TRIGGER

- São procedimentos armazenados no SGBD
- É um objeto no SGBD (são armazenados no DD)
- São tipos especiais de Procedures mas não podem receber parametros
- Estão vinculados a tabelas especificas
- Tamanho máximo do trigger – 32K
- Tomar cuidado para não criar recursividade dentro do trigger
- Dentro do corpo de um trigger pode ter comandos DML, estruturas de controle, chamada a outros procedimentos...
- Não se pode usar em um trigger: Comandos DDL, Controle de transação, COMMIT, ROLLBACK, SAVEPOINT

# TRIGGER

Os gatilhos de banco de dados são executados implicitamente quando uma instrução INSERT, UPDATE ou DELETE (instrução de acionamento) é emitida junto da tabela/view associada, independente do usuário conectado ou da aplicação usada.

**Observação:** Os gatilhos de banco de dados podem ser definidos em tabelas e views. Se uma operação DML for emitida em uma view, o gatilho INSTEAD OF define que ações ocorrerão.
Se essas ações incluírem operações DML em tabelas, então serão disparados quaisquer gatilhos na(s) tabela(s)-base.

# TRIGGER

Elementos fundamentais na elaboração de um trigger:

-Tempo do Trigger
-Tipo de Trigger
-Evento de Acionamento do Trigger

# TRIGGER

## Tempo do Trigger

Determina quando o trigger dispara em relação ao evento de acionamento. As opções são:

- ✓BEFORE: Executa o corpo do gatilho antes do evento DML de acionamento em uma tabela.

- ✓AFTER: Executa o corpo do gatilho após o evento DML de acionamento em uma tabela.

# TRIGGER

## Tempo do Trigger - Before

Gatilhos BEFORE
Usa-se este tipo de gatilho com freqüência nas seguintes situações:

• Quando a ação do gatilho deve determinar se a instrução de acionamento possui permissão para ser concluída. Essa situação permite eliminar o processamento desnecessário da instrução de acionamento e rollbacks eventuais nos casos em que surge uma exceção na ação de acionamento.

• Para derivar valores de coluna antes de concluir uma instrução INSERT ou UPDATE de acionamento

• Este tipo de trigger age analisando a alteração antes da verificação das eventuais constraints que existam na tabela.

# TRIGGER

## Tempo do Trigger - After

Gatilhos AFTER
Usa-se este tipo de gatilho com freqüência nas seguintes situações:

• Quando você deseja concluir a instrução de acionamento antes de executar a ação de acionamento

• Se um gatilho BEFORE já estiver presente e um gatilho AFTER puder executar ações diferentes na mesma instrução de acionamento

• Deseja-se realizar alguma ação tomando por base os dados já com a alteração consolidada e todas a restrições de integridades respeitadas.

# TRIGGER

## Tempo do Trigger – Instead Of

Gatilhos INSTEAD OF

Usa-se este tipo de gatilho para fornecer um modo transparente de modificação de views que não podem ser modificadas diretamente através de instruções SQL DML, porque a view não é modificada automaticamente.

É possível criar instruções INSERT, UPDATE e DELETE junto com a view, e o gatilho INSTEAD OF trabalhar invisível, em segundo plano, executando a ação codificada diretamente no corpo do gatilho nas tabelas subjacentes.

# TRIGGER

## Tipo do Trigger

Determina quantas vezes o corpo do trigger será executado. As opções são: Instrução e Linha

- ✓Instrução: Dispara o gatilho de instrução para todo o conjunto afetado pelo evento de acionamento, mesmo que nenhuma linha seja afetada.Os gatilhos de instrução são úteis se a ação do gatilho não depender dos dados das linhas afetados ou dos dados fornecidos pelo próprio evento de acionamento.
- ✓Linha: Dispara o gatilho para cada linha afetada pelo evento de acionamento. Se nenhuma linha for afetada pelo evento de acionamento, nenhum gatilho de linha será executado.
  Os gatilhos de linha são úteis se a ação do gatilho depender dos dados das linhas afetadas ou dos dados fornecidos pelo próprio evento de acionamento. É determinado pela instrução FOR EACH ROW.

# TRIGGER

## Evento de Acionamento do Trigger

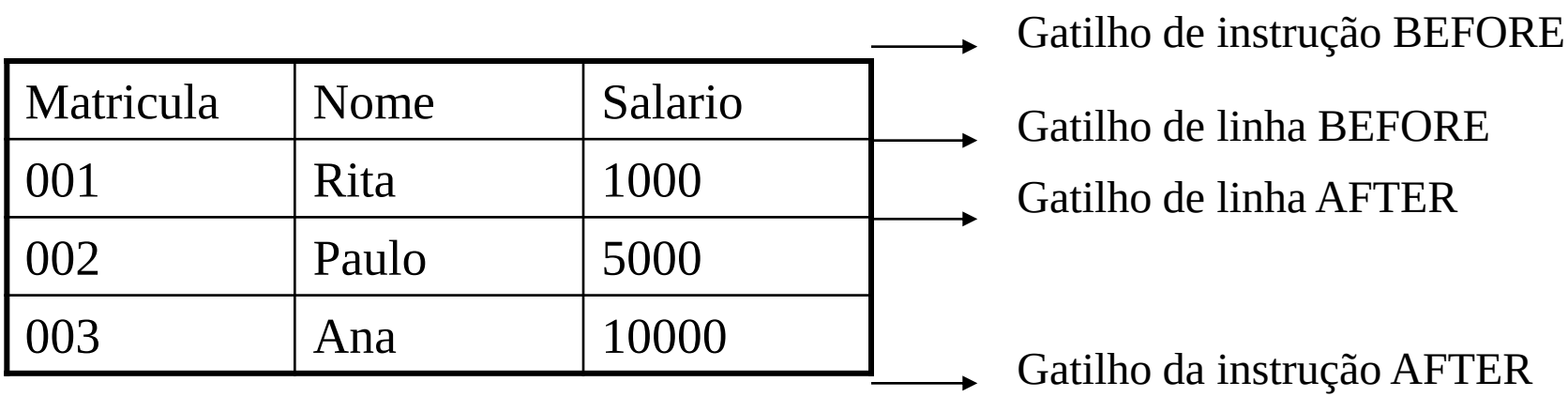
Que operações de manipulação de dados na tabela ou view disparam o trigger

- ✓Insert: Inserção
- ✓Update: Alteração. Permite especificar uma lista de colunas
- ✓Delete: Exclusão

WHEN Especifica a restrição de gatilho (Esse predicado condicional é avaliado para cada linha a fim de determinar se o corpo do gatilho é executado ou não.)

# TRIGGER

## Execução do Gatilho



# TRIGGER

## Predicados Condicionais

São usados para trigger acionados por mais de um evento de Acionamento.

- Inserting → Para trigger acionados pelo evento INSERT
- Deleting → Para trigger acionados pelo evento DELETE
- Updating → Para trigger acionados pelo evento UPDATE

# TRIGGER

## Memórias auxiliares

- Comparam as modificações ocorridas numa tabela
- Para guardar o dado deletado (DELETED - **OLD**)
- Para guardar o dado inserido (INSERTED - **NEW**)
- OLD é usada para UPDATE e DELETE
- NEW é usada para UPDATE e INSERT
- Estão disponíveis somente nos gatilhos ROW
- Devem ser precedidos por : (dois pontos) exceto se os qualificadores forem citados na clausula WHEN

# TRIGGER

Sintaxe

```
CREATE [OR REPLACE] TRIGGER nome_gatilho
        tempo
                evento1 [OR evento2 OR evento3]
                        ON nome_tabela
corpo_gatilho
```

# TRIGGER

Sintaxe

```
CREATE [OR REPLACE] TRIGGER nome_gatilho
        {BEFORE|AFTER|INSTEAD OF}
                evento1 [OR evento2 OR evento3]
                ON {nome_tabela | nome_view}
[REFERENCING OLD AS antigo | NEW AS novo]
[FOR EACH ROW]
corpo_gatilho
```

# TRIGGER

## Exemplo1: Trigger Before

Objetivo do Trigger: Validar se o salário é maior que 5000 somente para pessoas cargo de Gerencia ou Presidência

```
CREATE OR REPLACE TRIGGER TRIGGER_BEFORE
            BEFORE INSERT OR UPDATE OF sal ON emp
            FOR EACH ROW
BEGIN
  IF NOT (:NEW.CARGO IN (‘GERENTE' , 'PRESIDENTE'))
    AND :NEW.SALARIO > 5000
   THEN
     RAISE_APPLICATION_ERROR
            (-20202, 'EMPREGADO NÃO PODE GANHAR TANTO!');
  END IF;
END;
```

# TRIGGER

## Exemplo2: Trigger Before

Objetivo: Valida se o cadastramento do empregado está ocorrendo dentro do horário de expediente

```
CREATE OR REPLACE TRIGGER TRIGGER_BEFORE1
     BEFORE INSERT ON emp
BEGIN
   IF (TO_CHAR (sysdate,'DY') IN ('SAT','SUN')) OR
      (TO_CHAR(sysdate,'HH24') NOT BETWEEN
               '08' AND '18')
               THEN RAISE_APPLICATION_ERROR (-20500,
               'Só é possível incluir em EMP durante expediente.');
  END IF;
END;
```

TRIGGER

# Exemplo3: Trigger After

Objetivo: Salva os dados antes e depois de qualquer atualização em uma tabela de auditoria

```
CREATE OR REPLACE TRIGGER auditoria_emp
 AFTER DELETE OR INSERT OR UPDATE ON emp
 FOR EACH ROW
 BEGIN
 INSERT INTO audit_emp (usuario, hora_corrente, matricula, nome_antigo,
            nome_novo, cargo_antigo, cargo_novo, salario_antigo, salario_novo)
 VALUES (USER, SYSDATE, :OLD.empno, :OLD.ename,
             :NEW.ename, :OLD.job, :NEW.job, :OLD.sal, :NEW.sal );
 END;
```