# Banco de Dados

Aparecido Vilela Junior

# STORED SUBPROGRAM

# STORED SUBPROGRAM

- O objetivo de armazenamento uma procedure ou function no banco de dados é permitir que a rotina seja compartilhada por diversas aplicações.
- A forma de criação de uma rotina no banco de dados é semelhante à sintaxe para criação de uma rotina dentro de um bloco principal.
- A diferença está na cláusula CREATE, que estabelece o armazenamento da rotina no banco de dados

# STORED SUBPROGRAM

- Como vantagem temos:
  - A produtividade pode ser incrementada à medida que uma determinada rotina pode ser compartilhada por diversas aplicações, reduzindo a redundância de codificação
  - A performance pode ser incrementada uma vez que as rotinas armazenadas no banco de dados podem diminuir a necessidade de tráfego pela rede.
  - Em relação a segurança o DBA pode restringir o acesso direto a determinadas tabelas e autorizar a utilização apenas de rotinas que façam acesso.

# STORED SUBPROGRAM

```
CREATE OR REPLACE FUNCTION NOME(PDEPTO IN NUMBER) RETURN
    VARCHAR2 IS
  NOME      VARCHAR2(100);
BEGIN
  SELECT DNAME INTO NOME
    FROM DEPT
   WHERE DEPTNO = PDEPTO;
  RETURN NOME;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    NOME := 'DEPARTAMENTO INEXISTENTE';
    RETURN NOME;
END;
/
```

# Parâmetros

- Os Subprogramas recebem informações dos blocos que os acionaram através de parâmetros.
- Esses parâmetros são ditos formais.

# Modos dos Parâmetros

- Podem ser definidos de três formas:
  - IN: Indica um parâmetro de Entrada. Funciona como uma constante. É o default.
  - OUT: Indica um parâmetro de saída. Funciona como uma variável local.
  - IN OUT: Parâmetro usado simultaneamente como entrada e saída.

# Parâmetros

```
DECLARE
 VINSS        NUMBER := 2000;
 SALARIO      NUMBER := 3000;
 MSG          VARCHAR2(200);
 PROCEDURE INSS(SAL IN NUMBER, VALOR OUT NUMBER) IS
 BEGIN
  IF VALOR >= 1000 THEN
    VALOR := 1000;
  ELSE
    VALOR := SAL * 0.8;
  END IF;
 END INSS;
BEGIN
 INSS(3000, VINSS);
 INSS(SALARIO, VINSS);
 DBMS_OUTPUT.PUT_LINE('Valor do Inss: ' || to_char(VINSS));
END;
/
```

# OVERLOADING

- A PL/SQL permite a definição de dois subprogramas com o mesmo nome, desde que seus parâmetros formais sejam diferentes em número, ordem ou tipo familiar (por ex, Integer e Real são tipos diferentes, porém pertencem ao mesmo grupo familiar).
- Essa característica é chamada de Overloading.

# OVERLOADING

- Restrições:
  - Esta técnica só se aplica a subprogramas definidos em pacotes.
  - Além disso, não podemos "overload" dois subprogramas se seus parâmetros diferirem apenas em relação aos nomes ou aos modos de parâmetros.
  - Não podemos "overload" duas funções em que a única diferença entre elas seja o tipo de retorno (mesmo de familiares diferentes)

# OVERLOADING

```
DECLARE
  MSG          VARCHAR2(100);
  VINSS        NUMBER;
  PROCEDURE CALC (TIPO IN VARCHAR2 := 'P', VALOR IN NUMBER := 0.10) IS
  BEGIN
    MSG := MSG || 'PRIMEIRA CALC; ';
    IF UPPER(TIPO) = 'P' THEN
      UPDATE EMP
        SET SAL = SAL * VALOR + SAL;
    ELSE
      UPDATE EMP
        SET SAL = (SAL * VALOR) / 100 + SAL;
    END IF;
  END CALC;
  PROCEDURE CALC (SAL IN NUMBER := 2000, VALOR OUT NUMBER) IS
  BEGIN
    VALOR := SAL * 0.08;
    MSG := MSG || 'SEGUNDA CALC; ';
  END CALC;
BEGIN
  MSG := '';
  CALC;
  CALC('M', 30);
```

# PRIVILÉGIOS EM ROTINAS ARMAZENADAS

- Normalmente uma rotina armazenada no banco de dados (e métodos de objetos) executa com o privilégio de quem definiu a rotina (ou método) e não com o privilégio de quem aciona a rotina

# AUTHID CURRENT_USER

- A cláusula AuthId Current_User vem mudar essa situação.
- Quando uma função for criada com essa cláusula o Oracle passará a considerar tanto o schema quanto os privilégios do usuário que acionar a rotina (e não daquele dono da rotina) que executá-la.

# Referências Externas

- Os privilégios do executor da rotina são verificados a tempo de execução e as referências externas são resolvidas no schema deste executor, presentes em:
  - Comandos que manipulam os dados
  - Comandos para controle de transações
  - Comandos para controle de Cursores
  - Comandos de SQL Dinâmicos

# Funções para criação de Índices

- Os índices podem ser baseados em funções (e expressões) e não apenas baseados em colunas da tabela.
- Um índice baseado em função pré-calcula o valor da função (ou expressão) para todas as linhas da tabela e o armazena no índice.
- A utilização desta característica pode trazer significativos ganhos de performance.

# Funções para criação de Índices

- CREATE INDEX IND_UPPER ON FUNC (UPPER(NM_SOBRENOME));

- CREATE INDEX IND_SOMA ON EMP (SAL + COMM);

# Funções para criação de Índices

```
CREATE OR REPLACE FUNCTION SOMA(VALOR1 IN NUMBER, VALOR2 IN
    NUMBER)
      RETURN NUMBER DETERMINISTIC IS
BEGIN
  RETURN (NVL(VALOR1,-200) + NVL(VALOR2,-500));
END;
/
CREATE INDEX IND_SOMA_FUNC ON EMP(SOMA(SAL, COMM))
/
```

- Aqui foi criado um índice baseado em uma função definida pelo usuário. A cláusula Deterministic indica que se repetirmos os mesmos parâmetros para a função ela produzirá o mesmo resultado.

# PRÁTICA

- 1) Crie uma função armazenada no banco de dados que retorne o nome do departamento, cujo código deve ser passado como parâmetro. Caso o departamento não exista deve retornar 'Departamento Inexistente'
- 2) Crie uma função ou procedure na base de dados que determine o departamento que possui menos funcionários
- 3) Crie uma procedure na base de dados que receba como parâmetro um cargo e um código de departamento e determine o nome e o salário correspondentes a estes parâmetros. Se existir mais de um, escolha o funcionário mais novo. Se não existir nenhum retorne NULL
- 4) Faça uma rotina que receba como parâmetro um cargo e determine o nome do funcionário mais velho com aquele cargo. Se não houver ninguém, o programa deve abortar com o erro Oracle correspondente.

# Funciona ?

```
create or replace procedure AcheErro
is
valor     number(2) := 30;
v2          number(2) := 40;
begin
declare  /* Aqui começa o Segundo Bloco */
valor     varchar2(15) := 'Agora é Texto';
v3                    number;
begin
valor := 'Outro Texto';
v2 := v2+1;
end;
valor := valor + 1;
v2 := v2 +1;
v3 :=4;
  end;
```

# Prática

- Criar uma função que traga o Salário do Empregado, passando o código e retornando o salário, se não exisitir trazer 0.
- Crie um usuário e dê acesso para executar essa função.

# Prática II

- Crie uma tabela nova (func) no schema SCOTT, usando como referência a tabela EMP.

- Crie uma nova função (SALARIO2), mas que somente quem possa acessar tenha o mesmo privilégio do usuário que criou.

# Estrutura base

Pessoa(cd_pessoa, nm_pessoa,bo_bloqueado, vl_limite, vl_utilizado);

Endereco(cd_endereco,cd_pessoa, endereco, uf,bo_principal);

Nota(nr_nota, vl_nota, cd_pessoa, cd_endereco, impostos, frete);

Contareceber(nr_nota, vl_receber, dt_vencimento)

BXContareceber(nr_nota, nr_bx, dt_baixa, vl_baixa, vl_juros);

Pessoa x endereco (cd_pessoa);

Notaxpessoa(cd_pessoa);

Notaxendereco(cd_pessoa,cd_endereco);

Notaxcontareceber(nr_nota);

Contareceberxbxcontareceber(nr_nota, nr_bx)

# Functions

1. Cria uma função que retorne o endereço principal de uma pessoa;
2. Crie uma função que retorne quanto um cliente comprou no mês (passado por parâmetro);
3. Crie uma função que retorne a data da última compra de um cliente;

# Functions

4. Crie uma função que retorne qual é o cliente que mais comprou, qual fez a maior compra;
5. Crie uma procedure que imprima os seguintes dados:
   - Data da maior compra, Valor da maior compra, Qt. de Compra, Média de compra por mês.
   - Deve receber como entrada um período a ser analisado;
   - Deve usar funções dentro da procedure para realizar estes cálculos;