PROCEDURES, FUNÇÕES, PL/SQL, CURSORES, VIEW, ÍNDICE e SEQUÊNCIA

Prof. Edson Thizon

Stored procedure

- Uma Stored procedure é um grupo de comandos SQL e PL/SQL que executam determinada tarefa e podem ser invocados por um nome.

PL/SQL

- A linguagem PL/SQL é uma extensão da linguagem SQL que contém recursos das linguagens mais modernas. Permite construir blocos de comandos SQL para acesso e manipulação de base de dados.

Procedimentos e Funções

- ***Benefícios***:
  - permitem mover regras de negócios de aplicativos para o banco de dados, tornando-as independentes de aplicativos;
  - fornecem um mecanismo eficiente de reutilização de código;
  - aumentam a performance de aplicativos cliente-servidor;

Procedimento (procedure)

Sintaxe

```
CREATE [OR REPLACE] PROCEDURE [user.]procedure
  [ (argumento [IN |OUT|IN OUT] datatype
  [,(argumento [IN |OUT|IN OUT] datatype] ...)]
{IS/AS}

BLOCO;
```

Procedure

CREATE PROCEDURE *nome da procedure* (*parâmetro modo tipo*)
IS/AS *declarações de variáveis* BLOCO PL/SQL

parâmetro – nome da variável PL/SQL que será passado para a procedure.
modo – IN entrada
OUT saída

tipo – se o parâmetro é número, caracter, data ...
declaração de variáveis – nome, tamanho e tipo das variáveis que eu utilizarei no bloco PL/SQL

Procedure

Exemplo:

```
CREATE OR REPLACE PROCEDURE consulta
   (v_cod_empregado in emp.empno%type,
   v_nome_empregado out emp.ename%type,
   v_salario_empregado out emp.sal%type,
   v_com_empregado out emp.comm%type)
IS
BEGIN
   SELECT ename, sal, comm
   INTO v_nome_empregado, v_salario_empregado,
   v_com_empregado
   FROM emp
   WHERE empno = v_cod_empregado;
END consulta;
```

Chamando uma procedure

- As procedures podem ser chamadas de diversas formas:
  - Vamos supor que nós tenhamos uma procedure chamada proc_teste.
- Através do SQL*Plus:

  SQL> EXECUTE proc_teste;
- Através de um bloco (ex: procedure ou function):

```
BEGIN
   proc_teste;
END;
```

Procedure

Exemplo:

```
CREATE OR REPLACE PROCEDURE novo_periodo
   (p_cod_periodo in number,
   p_data_incial in date,
   p_data_final in date)
IS
BEGIN
   insert into periodo_letivo values(p_cod_periodo, p_data_incial,
                                     p_data final);
END;
```

Função (function)

Sintaxe

```
CREATE [OR REPLACE] FUNCTION  [user.]function
[(argumento IN datatype
[,argumento IN datatype]...)]
RETURN datatype
{IS/AS}

BLOCO;
```

Function

CREATE FUNCTION *nome da função* (*parâmetro modo tipo*)
RETURN *tipo*
IS/AS *declarações de variáveis*
BLOCO PL/SQL.

A diferença entre procedure e função é que a função sempre irá retornar um valor para o ambiente chamador.

Function

Exemplo

```
CREATE OR REPLACE FUNCTION salario
   (v_empno in emp.empno%type)
RETURN number
IS
   v_emp_sal emp.sal%type := 0;
BEGIN
   SELECT sal INTO v_emp_sal
   FROM emp
   WHERE empno = v_empno;
   RETURN (v_emp_sal);
END salario;
```

Function

Exemplo

```
CREATE OR REPLACE FUNCTION minuto
(dataim DATE, datafm DATE)
RETURN NUMBER
IS
hora NUMBER(8):=0;
minu NUMBER(8):=0;
minutoh NUMBER(8):=0;
BEGIN
        hora := TRUNC((datafm - dataim)*24);
        minu := TRUNC(((datafm - dataim)*24-
                TRUNC((datafm - dataim)*24))*60);
        minutoh := (hora*60)+minu;
RETURN minutoh;
END;
```

Compilando e Removendo

- Tanto os procedimentos quanto as funções são compiladas no instante do comando CREATE, mas se nós necessitarmos compilá-las explicitamente, então os comandos são:
  - ALTER PROCEDURE *nome da procedure* COMPILE;
  - ALTER FUNCTION *nome da função* COMPILE;
- Quando nós quisermos remover esses objetos do Banco, os comandos são:
  - DROP PROCEDURE *nome da procedure*;
  - DROP FUNCTION *nome da função*;

PL/SQL

- A PL/SQL é uma linguagem procedural da Oracle que estende a SQL com comandos que permitem a criação de procedimentos de programação.
- A linguagem permite a declaração de constantes, variáveis, subprogramas (procedures e funções), que favorecem a estruturação de código, e possui mecanismos para controle de erros de execução.

Vantagens

- Suporte para SQL (DML e Transação);
- Performance (tráfego rede);
- Portabilidade (SO e plataforma);
- Produtividade (batch, relatórios, etc).

Características

- cada comando SQL deve terminar com um ponto e vírgula, exceto BEGIN, DECLARE e EXCEPTION;
- um bloco PL/SQL não é uma unidade de transação. Os comandos COMMIT e ROLLBACK devem ser usados conforme a necessidade da aplicação;
- Comandos DDL não são permitidos;
- Comandos SELECT que não retornam apenas uma linha causam uma EXCEPTION que pode ser tratada pelo usuário;
- Comandos DML podem processar várias linhas ao mesmo tempo;
- Se um erro não previsto for encontrado na execução de um bloco PL/SQL, então o Oracle emitirá uma mensagem indicando o erro e o código do erro ocorrido e em qual linha ocorreu o erro.

ESTRUTURA

A PL/SQL é estruturada em blocos. Cada bloco pode conter outros blocos. Em cada um desses blocos, podemos declarar variáveis que deixam de existir quando o bloco termina

DECLARE - Opcional
Variaveis, cursores, exceptions definidas pelo usuário

BEGIN - Obrigatório
- SQL
- PL/SQL

EXCEPTION – Opcional
Ações que são executadas quando ocorem os erros

END – obrigatório

BLOCO PL/SQL COM SUB-BLOCO

```
DECLARE
      DEFINIÇÃO DE VARIÁVEIS
BEGIN
      COMANDOS
      DECLARE
            DEFINIÇÃO DE VARIÁVEIS
      BEGIN
            COMANDOS
      EXCEPTION
            TRATAMENTO DE ERROS
       END;
END;
```

Exemplo

```
Declare
      V_variavel varchar2(5);
Begin
      Select nome_coluna Into v_variável
      From table_name;
Exeption
      When exception_name Then
      .....
End;
```

Obs: Sempre coloque um (;) no fim de SQL ou um PL/SQL.

COMANDOS SQL

SELECT,
INSERT,
UPDADE,
DELETE,
ROLLBACK,
COMMIT,
SAVEPOINT

DATATYPES MAIS UTILIZADOS

CHAR
VARCHAR2
INTERGER
NUMBER
DATE
BOOLEAN

```
Declare
      V_data            date;
      V_deptno      number(2) := 10;
      V_location    varchar2(13) := 'Atlanta';
      V_comm        contant   number :=1400;
```

EXEMPLO

```
DECLARE

      NOME                CHAR(30);
      SALARIO       NUMBER(11,2);
      DEPART        NUNBER(4);
      DTNASC        DATE;
      SIM           BOOLEAN;
      CONT          NUMBER(6) :=0;
       PERC      CONSTANT NUMBER(4,2):= 36.00;
```

%Type

O atributo %TYPE

Declara a variável de acordo com uma coluna definida no Banco de Dados;

Exemplo

```
      V_ename           emp.ename%Type;
      V_balance         number(7,2);
      V_min_balance     v_balance%Type  :=10;
```

FUNÇÕES UTILIZADAS EM PL/SQL

Podemos contar com o uso de funções de Caracteres, Numéricas, Data, Conversão, dentre outras.
Exemplo:

```
Declare
        Cargo_atual  char(10);
Begin
        Select upper(substr(cargo,1,10)) into
cargo_atual
        from funcionario
        where cd_func = 2150;
End;
```

EM COMANDOS PL/SQL PODEMOS UTILIZAR AS SEGUINTES FUNÇÕES:

FUNÇÕES DE ERRO

sqlerrm, sqlcode

FUNÇÕES DE CARACTERES

ascii, chr, initcap, length, lower, lpad, rpad, ltrim, rtrim, substr, upper.

FUNÇÕES NUMÉRICAS

abs, mod, round, trunc, sqrt.....

CONTROLE DE FLUXO

COMANDO IF

1. IF <condição> THEN
   <comandos>
   END IF;

2. IF <condição> THEN
   <comandos>
   ELSE
   <comandos>
   END IF;

Comando If

3. IF <condição> THEN
   <comandos>
   ELSIF <condição> THEN
   <comandos>
   END IF;

4. IF <condição> THEN
   <comandos>
   ELSIF <condição> THEN
   <comandos>
   ELSE
   <comandos>
   END IF;

Comando If

5. IF <condição> THEN
   IF <condição> THEN
   <comandos>
   END IF;
   END IF;

```
SQL> VARIABLE MSG VARCHAR2(100);
SQL> DECLARE
        VALOR NUMBER(7,2) := &VAL;
BEGIN
        IF VALOR > 0 THEN
        :MSG := 'Valor maior que zero';
        ELSIF VALOR = 0 THEN
        :MSG := 'Valor igual a zero';
        ELSE
        :MSG := 'Valor menor que zero';
        END IF;
END;
SQL>Entre o valor para val: 5
antigo   2:  VALOR NUMBER(7,2) := &VAL;
novo   2:  VALOR NUMBER(7,2) := 5;

Procedimento PL/SQL concluído com sucesso.

SQL> print msg
MSG
----------------------------------------------
Valor maior que zero
```

COMANDOS DE REPETIÇÃO

SÃO UTILIZADOS PARA EXECUTAR REPETIDAMENTE O CÓDIGO ESCRITO DENTRO DELES.

- **LOOP**
- **FOR LOOPs**
- **WHILE LOOP**
- **CURSOR FOR LOOPs**

EXEMPLO COMANDO LOOP

```
SQL> SET SERVEROUT ON ; (ATIVA O DBMS)
SQL> DECLARE
        X       NUMBER := 0;
        COUNTER NUMBER := 0;
BEGIN
  LOOP
        X := X + 1000;
        COUNTER := COUNTER + 1;
        IF COUNTER > 4 THEN
           EXIT; (sai do loop)
        END IF;
        :MSG := X ||' '|| COUNTER || 'LOOP';
  END LOOP;
END;
```

EXEMPLO COMANDO FOR .. LOOP

```
DECLARE
 A,B     NUMBER(3):= 0;
 BEGIN
      FOR A IN 1..25 LOOP
            B:= B + 1;
       DBMS_OUTPUT.PUT_LINE('LOOP1 - '||B);
       END LOOP;
 END;
```

EXEMPLO COMANDO WHILE .. LOOP

```
SQL> CREATE TABLE TESTE
        (X NUMBER(3), Y VARCHAR2(30), K DATE);
SQL> DECLARE
          X NUMBER(3);
          Y VARCHAR2(30);
          K DATE;
          J NUMBER(3);
        BEGIN
          X:= 0;
          WHILE X<= 100 LOOP
            K:= SYSDATE-X;
            Y := 30;
            INSERT INTO TESTE VALUES  (X,Y,K);
            X := X + 1;
          END LOOP;
          COMMIT;
 END;
SQL> SELECT * FROM TESTE;
```

CURSOR

CURSOR < nome-cursor > [parâmetro tipo,...] **IS**

< comando select >

EXEMPLO:

```
CURSOR MEU_CURSOR IS
          SELECT ENAME, EMPNO, SAL
            FROM EMP
            ORDER BY SAL DESC;
```

OPEN CURSOR / FETCH CURSOR

```
DECLARE
        CURSOR C1 IS
        SELECT ENAME, EMPNO, SAL FROM EMP
        ORDER BY SAL DESC;
        MY_ENAME CHAR(10);
        MY_EMPNO NUMBER (4);
        MY_SAL   NUMBER (7,2);
BEGIN
        OPEN C1;
        FOR I IN 1..100 LOOP
                FETCH C1 INTO MY_ENAME, MY_EMPNO,
                                        MY_SAL;
                EXIT WHEN C1%NOTFOUND;

                DBMS_OUTPUT.PUT_LINE (MY_EMPNO
                        || ' ' || MY_ENAME|| ' ' || MY_SAL);
        END LOOP;
        CLOSE C1;
END;
```

FOR .. LOOP - CURSOR

```
DECLARE
        CURSOR C1 IS
        SELECT ENAME, EMPNO, SAL FROM EMP
        ORDER BY SAL DESC;

BEGIN
        FOR R1 IN C1 LOOP
          DBMS_OUTPUT.PUT_LINE (R1. EMPNO
                        || ' ' || R1. ENAME|| ' ' || R1. SAL);
        END LOOP;
END;
```

EXCEÇÕES

SÃO USADAS NO PL/SQL PARA LIDAR COM QUAISQUER ERROS QUE OCORRAM DURANTE A EXECUÇÃO DE UM BLOCO.
HÁ DOIS TIPOS DE EXECEÇÕES, AS DEFINIDAS INTERNAMENTE PELA PL/SQL E AS DEFINIDAS PELO USUÁRIO.

NESTA PARTE VEREMOS APENAS ALGUMAS DELAS

SINTAXE:

EXCEPTION

```
WHEN <nome-exeception> THEN
        <comandos>;
 WHEN <nome-execetion> THEN
        <comandos>;
```

EXEMPLO EXCEPTION

NO_DATA_FOUND - Quando um select não retorna nenhuma linha
TOO_MANY_ROWS - Quando um select retorna mais de uma linha
OTHERS - Qualquer tipo de erro

```
DECLARE
        NOME   CHAR(15);
        CARGO CHAR(10);

BEGIN
        SELECT ENAME, JOB INTO NOME, CARGO
        FROM EMP
        WHERE ENAME = 'KONG';

EXCEPTION
        WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('REGISTRO INEXISTENTE  '||SYSDATE);

         WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE(' MUITOS REGISTROS '||SYSDATE);

        WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE(' OUTRO ERRO QUALQUER '||SYSDATE);

END;
```

```
BEGIN
        SELECT ENAME, JOB INTO NOME, CARGO
        FROM EMP
        WHERE DEPTNO = 20;

EXCEPTION
        WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('REGISTRO INEXISTENTE  '||SYSDATE);

         WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE(' MUITOS REGISTROS '||SYSDATE);

        WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE(' OUTRO ERRO QUALQUER '||SYSDATE);

END;
```

RETORNANDO ERROS

SQLERR - Retorna o número do erro
SQLERRM – Retorna o número e a descrição do erro

Exemplo

```
BEGIN
  INSERT INTO DEPT VALUES  (10,'COMP','XXX');
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
     DBMS_OUTPUT.PUT_LINE('ERRO - '||SQLERRM);
END;
```

VIEW (visão)

- Uma view é uma apresentação customizada de dados em uma ou mais tabelas.
- Uma view funciona como uma “Tabela Virtual”, permitindo relacionamentos e combinações de várias tabelas e Views, chamadas de Tabelas Básicas.
- A view pode mostrar toda uma tabela ou apenas parte dela.
- Existem várias vantagens de trabalhar com Views:
  - Podem ser criadas views de duas tabelas diferentes, simplificando a consulta aos dados.
  - Podem ser omitidas colunas de uma view, restringindo os dados presentes na tabela.
  - Podem ser feitas seleções dos dados contidos em uma tabela, mostrando apenas certos registros.
  - A organização de uma View funciona da mesma forma que uma tabela

VIEW (visão)

Sintaxe

```
CREATE [OR REPLACE] VIEW [user.]procedure
AS
SUBQUERY;
```

VIEW (visão)

Exemplo:

```
CREATE OR REPLACE VIEW VW_EMP_DEPT
AS
SELECT DNAME, ENAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
ORDER BY DNAME, ENAME;
```

```
SQL> SELECT * FROM VW_EMP_DEPT;

DNAME          ENAME
-------------- ----------
ACCOUNTING     CLARK
ACCOUNTING     KING
...
```

INDEX (Índices)

- Os índices têm como objetivo providenciar um acesso rápido às informações de um banco de dados.
- Os índices são criados para aumentar a velocidade de consulta ou classificação das informações em uma tabela ou view.
- Nos arquivos de índice existem referências a cada registro de uma tabela, ordenados por técnicas, tais como: Arquivos Seqüenciais, Árvore B+, Funções de Hash etc.
- O uso de índices degrada a performance de operações de inserção, deleção e atualização de tabelas, pois nestas operações os índices deverão ser atualizados.
- ***Para uma melhor performance, todas as colunas que são chaves (primária, estrangeira e única) devem ser indexadas.***

INDEX (índice)

Sintaxe

```
CREATE [UNIQUE] INDEX [user.]procedure
ON
Table (column {ASC/DESC});
```

INDEX (índice)

Exemplo:

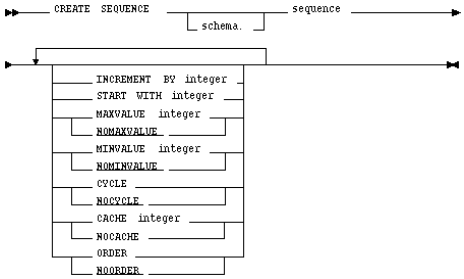
```
CREATE INDEX IDX_EMP_NOME
ON EMP (ENAME);
```

SEQUENCE (sequência)

- As sequences são objetos de um banco de dados que geram séries de números inteiros.
- As sequences são utilizadas para gerar identificadores únicos para cada registro de uma tabela.
- Você pode usar valores de uma sequence para assegurar a não existência de valores duplicados em uma coluna de um valor numérico.
- Possuem ótima performance em acessos simultâneos.

SEQUENCE

Sintaxe



Keywords and Parameters

schema
is the schema to contain the sequence. If you omit *schema*, Oracle7 creates the sequence in your own schema.

sequence
is the name of the sequence to be created.

INCREMENT BY
specifies the interval between sequence numbers. This integer value can be any positive or negative integer, but it cannot be 0. This value can have 28 or less digits. The absolute of this value must be less than the difference of MAXVALUE and MINVALUE. If this value is negative, then the sequence descends. If the increment is positive, then the sequence ascends. If you omit this clause, the interval defaults to 1.

MINVALUE
specifies the sequence's minimum value. This integer value can have 28 or less digits. MINVALUE must be less than or equal to START WITH and must be less than MAXVALUE.

NOMINVALUE
specifies a minimum value of 1 for an ascending sequence or $-(10^{26})$ for a descending sequence. The default is NOMINVALUE.

MAXVALUE
specifies the maximum value the sequence can generate. This integer value can have 28 or less digits. MAXVALUE must be equal to or less than START WITH and must be greater than MINVALUE.

NOMAXVALUE
specifies a maximum value of 10^{27} for an ascending sequence or -1 for a descending sequence. The default is NOMAXVALUE.

START WITH
specifies the first sequence number to be generated. You can use this option to start an ascending sequence at a value greater than its minimum or to start a descending sequence at a value less than its maximum. For ascending sequences, the default value is the sequence's minimum value. For descending sequences, the default value is the sequence's maximum value. This integer value can have 28 or less digits.

CYCLE
specifies that the sequence continues to generate values after reaching either its maximum or minimum value. After an ascending sequence reaches its maximum value, it generates its minimum value. After a descending sequence reaches its minimum, it generates its maximum.

NOCYCLE
specifies that the sequence cannot generate more values after reaching its maximum or minimum value. The default is NOCYCLE.

CACHE
specifies how many values of the sequence Oracle7 pre-allocates and keeps in memory for faster access.

NOCACHE
specifies that values of the sequence are not pre-allocated. If you omit both the CACHE parameter and the NOCACHE option, Oracle7 caches 20 sequence numbers by default.

SEQUENCE

Exemplo:

```
CREATE SEQUENCE SEQ_NUMEROEMP
INCREMENT BY 1
START WITH 1
MAXVALUE 9999999
MINVALUE 1
NOCYCLE
NOCACHE
ORDER;
```

Pega o próximo Valor da Sequence:

```
SQL> SELECT SEQ_NUMEROEMP.NEXTVAL FROM DUAL;

   NEXTVAL
----------
         1
```

Valor corrente da sequence:

```
SQL> SELECT SEQ_NUMEROEMP.CURRVAL FROM DUAL;

   CURRVAL
----------
         1
```

Referência Bibliográfica

- Documentação Oracle 7 e 8i.
- FERNANDES, Lúcia. **Oracle 9i** para desenvolvedores : Oracle developer 6i curso completo. Rio de Janeiro : Axcel Books, 2002. 1614 p.