Gerenciamento de Dados e Informação

Valéria Times
vct@cin.ufpe.br

Universidade Federal de Pernambuco
CIn.ufpe.br

PL/SQL

- **P**rocedural **L**anguage/**SQL**
  - Linguagem de programação sofisticada, utilizada para ter acesso a uma base de dados Oracle a partir de vários ambientes
    - Integrada no servidor da base de dados
    - Também disponível em algumas ferramentas cliente Oracle
    - A partir de aplicações desenvolvidas em outras linguagens
      - Aplicações Java utilizando JDBC
- O Modelo para a criação de PL/SQL é a linguagem ADA

2

PL/SQL

- Combina o poder e a flexibilidade de SQL com as estruturas de código de procedimentos encontradas nas linguagens de programação de 3a. geração
  - Estruturas de procedimento como
    - Variáveis e tipos (pré-definidos ou não)
    - Estruturas de controle (IF-THEN-ELSE e laços)
    - Procedimentos e funções
    - Tipos de objeto e métodos (Versão 8 em diante)

3

Elementos Básicos de PL

4

Variáveis

- Variáveis e Tipos
  - Utilizadas para transmitir informação entre programa PL/SQL e a base de dados
  - Localização de memória que pode ser lida ou ter valor armazenado a partir do programa PL/SQL
  - Não inicializadas recebem por default o valor NULL

5

Variáveis

- Identificadores (Nomes de variáveis)
  - Seqüência de até 30 caracteres
  - Inicia por letra
  - Os demais podem ser letras, dígitos, sublinhado e cifrões
  - Não são "case sensitive"
  - Não deve ser uma palavra reservada
  - Evitar usar nome de colunas da base de dados

6

Variáveis

- Tipos
  - Mesmos usados pelo Oracle

Numérico	BINARY_INTEGER inteiro de -2^{31} a 2^{31} NATURAL inteiro de 0 a 2^{31} POSITIVE inteiro de 1 a 2^{31} NUMBER(p,e) onde P é a precisão e E a escala (parte decimal)
Caractere	CHAR(N) onde N é o tamanho fixo da string. VARCHAR2(N) onde N é o tamanho máximo da string
Booleano	BOOLEAN onde os valores lógicos são TRUE ou FALSE
Data-Tempo	DATE não esquecer de usar apóstrofo ' '. Para fazer operações usar funções do Oracle para Date-Time

Registros

- Declaração e uso

```
CREATE TYPE nome IS RECORD (
  <campos> );
```

```
<identificador1> <tipo1>,
...
<identificadorn> <tipon>
```

Uso:

```
...
<variável> nome;
```

- Para declarar registro com mesmos tipos que uma tabela da base de dados

```
<variável> cliente%ROWTYPE;
```

Nome de tabela

Registros

- Exemplo – Um funcionário definido por nome, CPF e salário

```
CREATE TYPE Funcionario IS RECORD (
 nome  varchar2(30),
 CPF   varchar2(13),
 Salario  number(8,2));
```

- Uso

```
V_func Funcionario;
```


Tabelas

- Estrutura virtual análoga às Tabelas Relacionais
  - Só existe em memória principal, durante a execução do programa PL
  - Acesso só pode ocorrer através da indexação dos elementos da tabela
    - Utilizar tipo binary_integer para definir o índice da tabela
    - Não se pode usar SELECT, INSERT, etc., para acessar este tipo de estrutura

Tabelas

- Estrutura virtual análoga às Tabelas Relacionais (Cont.)
  - Define-se o tipo da tabela e depois uma variável deste tipo
  - Funcionam analogamente a matrizes em C

```
CREATE TYPE <tipo-tabela> IS TABLE OF <tipo-de-dado>
  INDEX BY  BINARY_INTEGER;
```


Tabelas

- Estrutura virtual análoga às Tabelas Relacionais (Cont.)
  - <tipo-de-dado> pode ser uma referência a um tipo escalar através de %TYPE

```
CREATE TYPE Tabela IS TABLE OF carro.modelo%TYPE
INDEX BY  BINARY_INTEGER;
```

Uso

```
v_modelo Tabela;
...
v_modelo(i)...
```

Acesso

Estrutura com uma coluna do tipo definido para o atributo MODELO da tabela CARRO

i deve ser do tipo binary_integer

Operadores

- Análogos, na sua maioria, aos das outras linguagens de programação
- Alguns exemplos
  - Aritméticos +, -, *, **, /
  - Atribuição :=
  - Diferente de < > ou ~=
  - Referência à base de dados @

13

Declaração e Inicialização de Variáveis

- Comentários

```
-- indica comentário de uma linha
/* indica comentário de mais de
uma linha */
```

- Data do sistema → SYSDATE

```
Ex.:
  data_saida DATE := SYSDATE;
```

14

Declaração e Inicialização de Variáveis

- Declaração de constante

```
desconto_padrao CONSTANT NUMBER(3,2) := 8.25;
```

Quantidade de Dígitos

Casas decimais

- Declaração de valor default

```
participante BOOLEAN DEFAULT TRUE;
```

- Declaração de variável com tipo de um atributo de tabela

```
<variavel> carro.modelo%TYPE;
```

15

Blocos

16

Blocos

- Várias instruções de SQL podem estar contidas em um único bloco de PL/SQL e serem enviadas como uma só unidade para o servidor de banco de dados
- Estrutura de blocos
  - Unidade básica
  - Todos os programas são construídos por blocos que podem ser encadeados entre si
  - Cada bloco executa uma unidade lógica de trabalho

17

Blocos

- Estrutura de blocos (Cont.)

```
DECLARE
 /* Seção para declarar variáveis, tipos,
       cursores e subprogramas locais */

BEGIN
/*Seção executável - comandos procedurais e SQL.
  É a única obrigatória */
EXCEPTION
  --Comandos de manipulação de erros
END;
```

18

Blocos

- Exemplo

```
DECLARE
  /* Declaração de variáveis que serão utilizadas  em
comandos SQL */
  v_telefone VARCHAR2(10) := '21268430';
  v_cpf VARCHAR2(12) := '123456789-34';
BEGIN
  -- Atualiza a tabela CLIENTE.
  UPDATE cliente
    SET telefone = v_telefone
    WHERE cpf = v_cpf;
```

Valores iniciais

Blocos

- Exemplo (Cont.)

```
EXCEPTION
/* Verifica se o registro foi encontrado. Em caso contrário,
insere na tabela como tupla nova. */
  IF SQL%NOTFOUND THEN INSERT INTO cliente
(cpf, telefone)
     VALUES (v_cpf, v_telefone);
  END IF;
END;
/
```

Símbolo para mandar executar o bloco

Blocos

- Blocos anônimos
  - Construídos de forma dinâmica e executados só uma vez

```
DECLARE
   <definições  de variáveis>
BEGIN
  <comandos>
EXCEPTION
<tratamento de erros de execução>
END;
```


Blocos

- Blocos nomeados
  - Blocos anônimos com um rótulo que fornece o nome do bloco

```
<<l_nome>>
DECLARE
   <definições  de variáveis>
BEGIN
  <comandos>
EXCEPTION
<tratamento de erros de execução>
END l_nome;
```


Escopo de Variável

- Definida no bloco → local
- Definida fora do bloco → global

```
DECLARE
sexo CHAR:='F';
...
BEGIN
...
END;
DECLARE
...
sexo CHAR:='M';
...
...sexo...   ?
END;
```

Aqui é M

Rótulo

```
<<global>>
DECLARE
sexo CHAR:='F';
...
BEGIN
...
END global;
DECLARE
...
sexo CHAR:='M';
...sexo...
...global.sexo...
END global;
```

Aqui é F

Controle de Processamento

Estruturas de Controle

- Comando IF-THEN-ELSE

```
IF <expressão booleana 1> THEN <instruções1>
[ELSIF <expressão booleana 2> THEN
<instruções2>]
[ELSE <instruções3>]
END IF;
```


Estruturas de Controle

- Exemplo
  - Se a média do estudante for maior ou igual a 7.0, colocar em situação 'Aprovado', se for menor que 5.0, colocar 'Reprovado'. Em caso contrário, 'Final'

```
IF media >= 7.0 THEN situacao := 'Aprovado';
ELSIF media < 5.0 THEN
situacao := 'Reprovado';
ELSE situacao := 'Final';
END IF;
```


Estruturas de Controle

- Comando CASE

```
CASE <seletor>
WHEN <valor1> THEN <instruções 1>;
...
WHEN <valorn> THEN <instruções n>;
[ELSE <instruções m>;]
END CASE;
```


Estruturas de Controle

- Exemplo
  - Se o valor de uma variável for 2, calcule o dobro; se for 5 some 15; para qualquer outro valor, calcule o triplo. Armazene o resultado em uma variável chamada valor

```
CASE i
WHEN 2 THEN valor := 2 * i;
WHEN 5 THEN valor := i + 15;
ELSE valor := i * 3 ;
END CASE;
```


Laços

- Permitem executar a mesma seqüência de instruções várias vezes
  - Laço simples

```
LOOP
<instruções>
END LOOP;
```

  - Executam infinitamente, a menos que seja colocada instrução de saída
    - EXIT [WHEN <condição>];

Laços

- Laço WHILE

```
WHILE <condição> LOOP
<instruções>
END LOOP;
```

  - Exemplo
    - Montar uma tabela virtual (em memória) com dois atributos cujo campo chave contém valores naturais e o outro atributo contém o quadrado do valor da chave, desde que menor que 30

Laços

- Laço WHILE (Cont.)

```
TYPE elemento IS RECORD (
chave INTEGER,
valor INTEGER);
TYPE tabela IS TABLE OF elemento
     INDEX BY BINARY_INTEGER;
i BINARY_INTEGER;
c tabela;
BEGIN
i := 1;
```


Laços

- Laço WHILE (Cont.)

```
WHILE i**2 < 30
LOOP
c(i).chave := i;
c(i).valor := i**2;
i := i+1;
END  LOOP;
END;
```


Laços

- Laço FOR

```
FOR <contador> IN [REVERSE] <inferior> .. <superior>
 LOOP
<instruções>
END LOOP;
```

- Exemplo
  - Armazenar na tabela exemp1(linha, valor) os elementos da tabela virtual C criada no exemplo anterior

Laços

- Laço FOR (Cont.)

```
FOR m IN 1..i-1 LOOP
INSERT INTO exemp1(linha, valor)
  VALUES(c(m).chave, c(m).valor);
END LOOP;
END;
```

O índice i da tabela virtual C saiu do laço anterior com uma unidade a mais no valor

Laços

- GOTO e Rótulos

```
...
GOTO rot1;
...
<<rot1>>
```

Rótulo

- Laços podem ser etiquetados para uso em EXIT

Recuperação de Dados do BD para Variáveis

Recuperação de Dados para Variável com SELECT

- Utilizar comando → SELECT ... INTO

```
SELECT <atributo(s)>  INTO <variável(is)>
 FROM <tabela(s)> WHERE... ;
```

Nunca use:

~~v := SELECT <atributo(s)> FROM <tabela(s)> WHERE... ;~~

- Considere
  - Número de <variável(is)> deve ser igual ao número de <atributo(s)>

Recuperação de Dados para Variável com SELECT

- Considere (Cont.)
  - Os tipos de cada <atributo> e da <variável> correspondente devem ser compatíveis
  - Deve ser recuperada uma única tupla
  - <variável(is)> devem ser declaradas
    - Ex.: Uma variável denominada qtdEmp

```
qtdEmp  NUMBER;
```


Recuperação de Dados para Variável com SELECT

- Exemplo: Armazenar em qtdEmp a quantidade de empregados de uma companhia

```
SELECT COUNT(*) INTO qtdEmp
FROM Empregado;
```


Saída de Dados

Saída de Dados

- Na interface de caracteres
  - Para permitir Output (Saída)

```
Set serveroutput on;
```

  - Comandos de saída

Escreve na mesma linha

```
dbms_output.put(' ... ') ou
dbms_output.put_line (' ... ')
```

Escreve e depois muda de linha

Parâmetros devem ser cadeias de caracteres

Saída de Dados

- Na interface de caracteres (Cont.)
  - A saída é uma cadeia de caracteres (string)
  - Funções Oracle para manipulação de strings

Função	Ação
UPPER(<string>);	Converte para maiúscula
RTRIM(<string>)	Remove espaços à direita
LENGTH(<string>)	Tamanho da string
LOWER(<string>)	Converte para minúscula
INSTR(<string$_1$>, <string$_2$>)	Informa a posição inicial da <string$_2$> em <string$_1$>
SUBSTR(<string>,m,n)	Sub-string das posições m a n
TO_CHAR(<valor>, [<formato>])	Converte data ou número em string

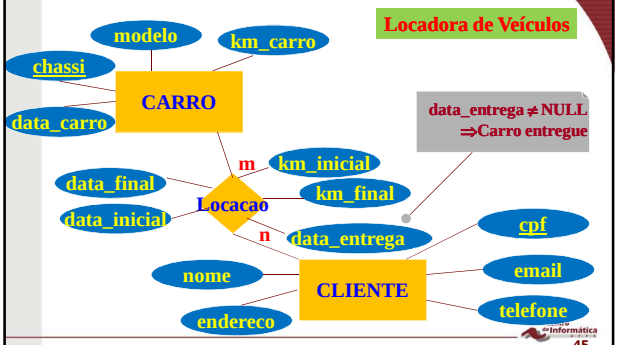
Saída de Dados

- Na interface de caracteres) (Cont.)
  - Operador Oracle para manipulação de strings

Operador	Ação
<string$_1$> || <string$_2$>	Concatena o <string$_2$> ao final do <string$_1$>

Modelo Exemplo

Exemplo – Modelo E-R



Tratamento de Exceções

Tratamento de Exceções

- Responde a erros de execução do programa
  - Exemplo: Para dado não encontrado

```
DECLARE
  v_chassi VARCHAR(20) := '235-456-YWR';
/* Variável alfanumérica inicializada com
235-456-YWR */
  v_modelo VARCHAR2(20);
/* Tamanho de variável string com no
   máximo 20 caracteres */
```


Tratamento de Exceções

- Exemplo (Cont.)

```
BEGIN
 /* Início da Execução recupera modelo do carro com chassi
    235-456-YWR */
 SELECT modelo
  INTO v_modelo
  FROM carro
  WHERE chassi = v_chassi;
```

Tratamento de Exceções

- Exemplo (Cont.)

```
EXCEPTION
 -- Seção de Tratamento de Exceção
 WHEN NO_DATA_FOUND THEN
  -- Manipula a condição de erro
  INSERT INTO log_table (info)
    VALUES ('Carro com Chassi 235-456-YWR não existe! ');
END;
/
```


Tratamento de Exceções

- Exceções pré-definidas pelo Oracle

Exceção	Significado
NO_DATA_FOUND	Não há tupla recuperada
TOO_MANY_ROWS	Excesso de tuplas recuperadas
INVALID_CURSOR	Erro de definição de cursor
ZERO_DIVIDE	Divisão por zero
DUP_VAL_ON_INDEX	Índice duplicado

- Para cada tipo de erro pode-se colocar um WHEN na seção EXCEPTION
- A opção WHEN OTHERS pode ser usada para tratar qualquer erro diferente dos listados

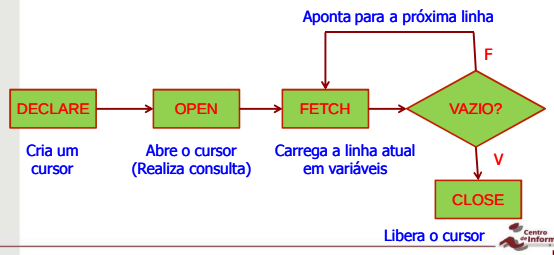
Cursores

Cursores

- Utilizados para processar várias linhas obtidas a partir da base de dados (a partir de uma instrução SELECT)
  - Programa pode percorrer o conjunto de linhas, devolver uma de cada vez e processar cada uma delas
- Podem ser explícitos ou implícitos
  - Declarados e gerenciados pelo programador
  - Declarados e gerenciados pelo Oracle

Cursores

- Cursores explícitos
  - Fluxo de controle



Cursores

- Utilização de Cursores explícitos
  - Declarar o cursor

```
CURSOR <nome> IS <comando_select>
```

  - Abrir o cursor para consulta

```
OPEN <nome>;
```

  - Extrair os resultados para variáveis PL/SQL

```
FETCH <nome> INTO <lista_de_variáveis>; ou
FETCH <nome> INTO  <registro>;
```

  - Fechar o cursor

```
CLOSE <nome>;
```

Cursores

- Verificação de propriedades de cursores explícitos

Propriedade	Significado
%rowcount	Quantidade de linhas afetadas pelo comando que gerou o cursor
%found	True → caso alguma linha tenha sido afetada
%notfound	True → caso não tenha sido afetada alguma linha
%isopen	True → caso o cursor esteja aberto

55

Cursores

- Exemplo: Cursor para manipulação de Carros

```
set serveroutput on;
DECLARE
  /* Variáveis de saída para guardar resultados da
consulta */
  v_chassi    carro.chassi%TYPE;
  v_data_carro  carro. data_carro%TYPE;
  v_km_carro    carro.km_carro%TYPE;
-- Limitar modelo usado na consulta
  v_modelo       carro.modelo%TYPE := 'Clio Sedan';
```

56

Cursores

- Exemplo (Cont.)

```
-- Declaração do Cursor
 CURSOR c_carro IS
   SELECT chassi, km_carro, data_carro
    FROM carro
    WHERE modelo = v_modelo;
BEGIN
 /* Preparar para futuro processamento dos dados – Abrir
o cursor*/
  OPEN c_carro;
```

57

Cursores

- Exemplo (Cont.)

```
LOOP
   /* Recupera cada tupla do cursor em variáveis PL/SQL
*/
   FETCH c_carro INTO v_chassi, v_km_carro,
v_data_carro;
/* Se não há mais tuplas para trazer, saída do laço */
   EXIT WHEN c_carro%NOTFOUND;
/*Processamento das tuplas para saída na interface de
caracteres*/
```

58

Cursores

- Exemplo (Cont.)

```
DBMS_OUTPUT.PUT_LINE('Carro: '|| ''||
TO_CHAR(v_chassi) || ' ' || TO_CHAR(v_km_carro) || ' '
|| TO_CHAR(v_data_carro));
  END LOOP;
-- Liberar recursos utilizados – Fechar o cursor
  CLOSE c_carro;
END;
/
```

59

Cursores

- Tipo registro em cursor – Listar dados de Clientes

```
DECLARE
CURSOR c_reg IS
SELECT cpf, nome FROM Cliente
WHERE endereco = 'Rio';
v_reg c_reg%ROWTYPE;

BEGIN
OPEN c_reg;
```

Definição de Variáveis

60

Cursores

- Tipo registro em cursor (Cont.)

```
LOOP
FETCH c_reg INTO v_reg;
EXIT WHEN c_reg%NOTFOUND;
DBMS_OUTPUT.PUT_LINE( v_reg.cpf ||
' '||v_reg.nome);
END LOOP;
CLOSE c_reg;
END;
/
```

Manipulação

Cursores

- Cursor com laços FOR

```
DECLARE
CURSOR c_reg IS
SELECT cpf, nome FROM Cliente
WHERE endereco = 'Rio';

BEGIN
FOR v_reg IN c_reg LOOP
DBMS_OUTPUT.PUT_LINE( v_reg.cpf ||
' '||v_reg.nome);
END LOOP;
END;
/
```

Forma mais reduzida de manipular cursores, pois:
- Faz uso implícito dos comandos OPEN, FETCH, EXIT e CLOSE
- Faz a declaração implícita da variável de tipo registo

Cursores

- Cursor com laços FOR (sem declaração)

```
DECLARE
...

BEGIN
FOR v_reg IN (SELECT cpf, nome FROM Cliente
WHERE endereco = 'Rio') LOOP
DBMS_OUTPUT.PUT_LINE( v_reg.cpf ||
' '||v_reg.nome);
END LOOP;
END;
/
```


Cursores

- Cursor com parâmetros

```
DECLARE
CURSOR c_reg (p_valor VARCHAR2) IS
SELECT cpf, nome FROM Cliente
WHERE endereco = p_valor;
v_reg  c_reg%ROWTYPE;

BEGIN
OPEN c_reg('Rio');
LOOP
FETCH c_reg INTO v_reg;
EXIT WHEN c_reg%NOTFOUND;
```


Cursores

- Cursor com parâmetros (Cont.)

```
DBMS_OUTPUT.PUT_LINE( v_reg.cpf ||
' '||v_reg.nome);
END LOOP;
CLOSE c_reg;
...
END;
/
```

Cada nova passagem de parâmetro exige um OPEN(parâmetro)/CLOSE

- Cursores implícitos
  - Utilizados para processar instruções INSERT, UPDATE, DELETE e SELECT.. INTO

Cursores

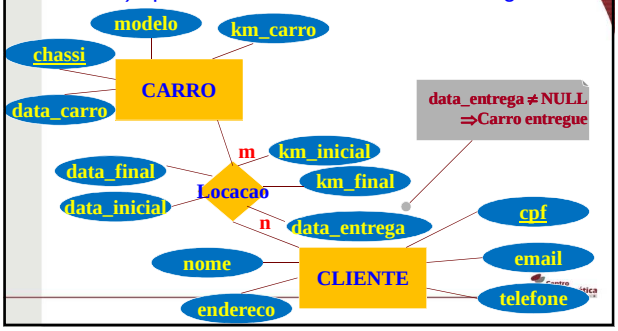
- Cursores implícitos (Cont.)
  - Exemplo

```
BEGIN
 UPDATE cliente
   SET email = 'nina@cin.ufpe.br'
   WHERE cpf = '324567876-18';
 /* Se o UPDATE não encontrar nenhuma tupla, inserir
       nova tupla na tabela */
IF SQL%NOTFOUND THEN
   INSERT INTO cliente (cpf, email) VALUES ('324567876-
18', 'nina@cin.ufpe.br');
 END IF;
END;
```

Cursor implícito

Exercício - Cursores

- Escreva um bloco para executar no Oracle, e imprimir na tela todos os clientes (cpf, nome e email) que não devolveram o carro alugado.



Exercício - Cursores

- Escreva um bloco para executar no Oracle, e imprimir na tela todos os clientes (cpf, nome e email) que não devolveram o carro alugado.

```
set serveroutput on;

DECLARE
  v_cpf      cliente.cpf%TYPE;
  v_nome     cliente. nome%TYPE;  /* Variáveis de saída */
  v_email    cliente.email%TYPE;

CURSOR  c_cliente  IS        /* Declaração do cursor*/
   SELECT  C.cpf,  C.nome,  C.email
      FROM  cliente  C,  locacao  L
      WHERE  L.cpf_cli  = C.cpf  AND
                 L.data_entrega = NULL;
```

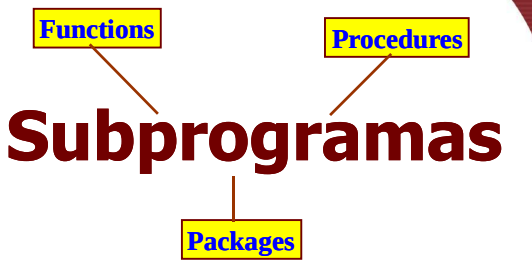
Exercício - Cursores

```
BEGIN
    OPEN  c_cliente;

    LOOP
        FETCH c_cliente INTO v_cpf, v_nome, v_email;
        EXIT WHEN  c_cliente%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('Cliente: ' ||
                    v_cpf || ' ' || v_nome || ' ' || v_email);
    END LOOP;

    CLOSE  c_cliente;
END;
/
```



Subprogramas

- **Procedimentos**, **funções** e **pacotes** que são armazenados na base de dados
- Em geral não são alterados depois de construídos e são executados muitas vezes
- São executados explicitamente, através de uma chamada a eles

Procedimentos

- Sintaxe

Indica parâmetro de entrada e saída

Indica parâmetro de entrada (*default*)

Indica parâmetro de saída

```
CREATE [OR REPLACE] PROCEDURE <nome>
 [(parâmetro [{IN | OUT | IN OUT}] tipo, ....)]
IS
[<definições  de variáveis> ]
BEGIN <corpo-do-procedimento>
END <nome>;
```

Observar que não se usa a Cláusula DECLARE

Procedimentos

- Exemplo: Procedimento para inserir um novo veículo na tabela Carro

```
CREATE OR REPLACE PROCEDURE InsereCarro (
p_chassi  carro.chassi%TYPE,
  p_modelo   carro.modelo%TYPE,
  p_km_carro carro.km_carro%TYPE,
  p_data_carro carro.data_carro %TYPE) AS
BEGIN
  -- Inserir nova tupla na tabela carro
INSERT INTO carro (chassi, modelo, km_carro,
data_carro)
   VALUES (p_chassi, p_modelo, p_km_carro,
p_data_carro);
```


Procedimentos

- Exemplo (Cont.)

```
COMMIT;
END InsereCarro;
/
```

- Para ser chamada em outros blocos PL/SQL

```
InsereCarro('235-456-YWR','Celta', 100,
to_date('15/05/2002','dd/mm/yyyy'));
```

- Para ser chamada na interface de caracteres

```
EXEC InsereCarro('235-456YWR','Celta', 100,
to_date('15/05/2002','dd/mm/yyyy'));
```


Funções

- São chamadas como parte de uma expressão, retornando um valor à expressão

```
CREATE [OR REPLACE ]FUNCTION <nome>
 [(parâmetro  tipo, ....)]
RETURN  <tipo-retorno> IS
BEGIN <corpo-da-função>
END <nome>;
```

Tipo do valor a ser retornado

- No corpo da função deve aparecer um RETURN para retornar um valor ao ambiente

```
RETURN <expressão>;
```


Funções

- Exemplo: Uma função para calcular o valor de uma nova chave a partir da maior existente no BD

Não especificar tamanho na definição do tipo de retorno

```
CREATE OR REPLACE FUNCTION calcula RETURN
VARCHAR2 IS
retorno VARCHAR2(20);
BEGIN
   select max(id) into retorno from categoria;
      retorno := retorno +1;
   RETURN retorno;
END calcula;
/
```


Funções

- Para ser chamada em outros blocos PL/SQL

```
v_valor := calcula;
```

Variável declarada no bloco PL

- Para ser chamada na interface de caracteres

```
SELECT calcula FROM dual;
```