Gerenciamento de Dados e Informação

PL

Fernando Fonseca
Ana Carolina
Robson Fidalgo

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

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)

Elementos Básicos de PL

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

Identificador
Tipo

Variáveis

- Identificadores (Nomes de variáveis)
  - Sequência 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

Variáveis

- Tipos
  - Mesmos usados pelo Oracle

Numérico	BINARY_INTEGER inteiro de -2^{31} - 1 a 2^{31} -1. NATURAL inteiro de 0 a 231 POSITIVE inteiro de 1 a 231 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 → %ROWTYPE

```
<variável> cliente%ROWTYPE;
```

Nome de tabela

Registros

- Exemplo 1 – 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 (só ocorre em memória principal) análoga às Tabelas Relacionais
  - Só existe durante a execução do programa PL
  - Acesso só pode ocorrer por meio 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

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

  - <tipo-de-dado> pode ser uma referência a um tipo escalar utilizando %TYPE

Tabelas

- Estrutura virtual análoga às Tabelas Relacionais (Cont.)
  - Exemplo 2: Definir uma tabela virtual para armazenar informações do tipo de dados do atributo modelo da tabela carro

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

Uso:

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

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

Acesso a elemento da tabela

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 ➔ @

Indica instância de BD distribuído

Declaração e Inicialização de Variáveis

- Comentários

```
-- indica comentário de uma linha

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

- Data do sistema → Função SYSDATE

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


Declaração e Inicialização de Variáveis

- Declaração de constante

Quantidade de Dígitos

Casas decimais

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

- Declaração de valor default

```
participante BOOLEAN DEFAULT TRUE;
```

- Declaração de variável com tipo de um atributo de tabela

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


Blocos

Blocos

- Permitem várias instruções de SQL contidas em um único bloco de PL/SQL que podem ser 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

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;
```

Escopo de Variável

- Onde sua definição é válida
  - Definida no bloco → local
  - Definida fora do bloco → global

Rótulo

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

END;
```

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

Aqui é F

Aqui é M

Blocos

- Exemplo 3: Criar um bloco para atualizar o atributo telefone da tabela Cliente com o valor definido no bloco, para o cliente com CPF informado

```
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 3 (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;
/
```

Válido se demais atributos de Cliente não definidos como NOT NULL e um dos atributos informados for chave primária

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;
/
```


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 4: 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 5: Se o valor de uma variável V for 2, calcule o dobro; se for 5 some 15; para qualquer outro valor, calcule o triplo. Armazene o resultado em uma variável chamada i

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


Laços

- Permitem executar a mesma sequencia 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 6: Montar uma tabela virtual (em memória) com dois atributos: id contém valores inteiros a partir de 1 e info contém o quadrado do valor de id, desde que menor que 30

Laços

- Exemplo 6 (Cont.)

```
DECLARE
TYPE elemento IS RECORD (
id INTEGER,
info INTEGER);
TYPE tabela IS TABLE OF elemento
    INDEX BY BINARY_INTEGER;
i BINARY_INTEGER;
C tabela;
BEGIN
i := 1;
```


Laços

- Exemplo 6 (Cont.)

```
WHILE i**2 < 30
LOOP
C(i).id := i;
C(i).info := i**2;
i := i+1;
END  LOOP;
END;
/
```


Laços

- Laço FOR

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

- Exemplo 7: No bloco anterior, acrescentar comandos para armazenar na tabela relacional exemp1(linha, valor) os elementos da tabela virtual C

Laços

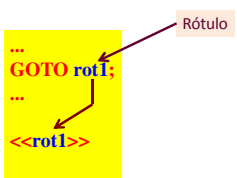
- Exemplo 7 (Cont.)

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

O índice i da tabela virtual C saiu do laço anterior com uma unidade a mais no valor, quando i**2 > 30

Laços

- GOTO e Rótulos



- Laços podem ser etiquetados para uso em EXIT

Recuperação de Dados do BD para Variáveis

Recuperação de Dados de Tabela para Armazenamento em Variável

- Utilizar comando → SELECT ... INTO

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

Nunca use:

~~<variável(is)> := 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;
```

  - Exemplo 8: 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 de servidor pleno
  - Para permitir Output (Saída)

```
Set serveroutput on;
```

  - Comandos de saída

Escreve e permanece na linha

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

Escreve e depois muda de linha

Parâmetros devem ser cadeias de caracteres

  - A saída é uma cadeia de caracteres (String)

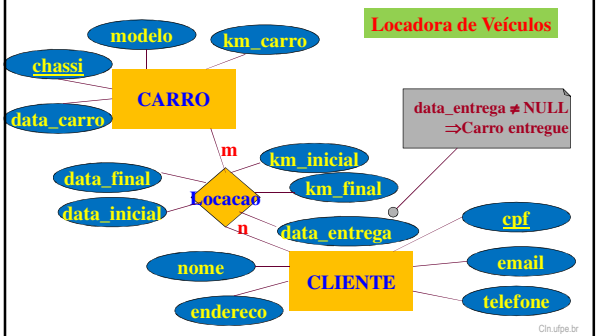
Saída de Dados

- 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>]) <string$_1$> || <string$_2$>	Converte data ou número em string 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 9: Informar modelo de carro com um dado Chassi. Para dado não encontrado, informar fato na tabela log_table (info)

```
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 9 (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 9 (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

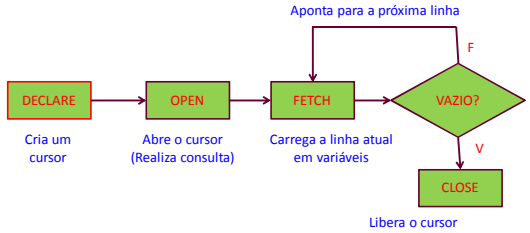
- Utilizados para processar várias linhas obtidas a partir da base de dados (por meio de uma instrução SELECT)
  - Programa pode percorrer o conjunto de linhas, devolver uma de cada vez e processar cada uma delas
- Podem ser explícitos ou implícitos

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 tuplas recuperadas pelo comando que gerou o cursor
%found	True → caso alguma tupla tenha sido recuperada
%notfound	True → caso não tenha sido recuperada alguma linha
%isopen	True → caso o cursor esteja aberto

Cursores

- Exemplo 10: 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';
```

Habilitar saída de dados no SQL+

Cursores

- Exemplo 10 (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;
```


Cursores

- Exemplo 10 (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*/
```


Cursores

- Exemplo 10 (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;
/
```


Cursores

- Tipo registro em cursor
  - Exemplo 11: Listar cpf e nome de Clientes com endereço igual a Rio

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

BEGIN
OPEN c_reg;
```


Cursores

- Exemplo 11 (Cont.)

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


Cursores

- Cursor com laços FOR
  - Exemplo 12: Listar nome e email de Clientes com endereço igual a Recife

```
DECLARE
CURSOR c_reg IS
SELECT nome, email FROM Cliente
WHERE endereco = 'Recife';
BEGIN
FOR v_reg IN c_reg LOOP
DBMS_OUTPUT.PUT_LINE( v_reg.nome ||
' '||v_reg.email);
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 do cursor)
  - Exemplo 13: Listar nome e telefone dos Clientes com endereço Salvador

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


Cursores

- Cursor com parâmetros
  - Exemplo 14: Listar cpf e nome de clientes com endereço igual a um dado valor

```
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

- Exemplo 14 (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

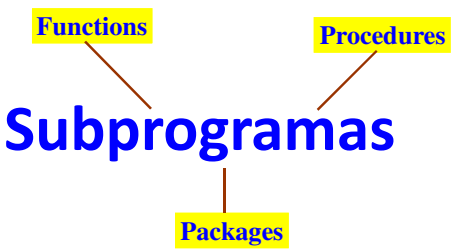
Cursores

- Cursores implícitos (Cont.)
  - Exemplo 15: Atualizar o e-mail do Cliente com CPF igual a 324567876-18 para 'nina@cin.ufpe.br'

```
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 ('3
18', 'nina@cin.ufpe.br');
 END IF;
END;
/
```

Cursor implícito

Desde que os demais atributos não tenham sido definidos como NOT NULL e um deles seja chave primária



Subprogramas

- **Procedimentos**, **funções** e **pacotes** que são criados e armazenados na base de dados
- Em geral não são alterados depois de construídos e são executados muitas vezes
- São executados explicitamente, por meio de uma chamada a eles
  - A forma da chamada depende do tipo de subprograma

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 16: 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
-- 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 16 (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 diretamente 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 comando RETURN para retornar um valor ao ambiente

```
RETURN <expressão>;
```


Funções

- Exemplo 17: 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 diretamente na interface de caracteres

```
SELECT calcula FROM dual;
```

PACKAGES

- Fornecem mecanismo para ampliar o poder da linguagem
- Os elementos do pacote podem aparecer em qualquer ordem, mas um elemento tem que ser declarado antes que seja referenciado
- Na definição do pacote só é apresentada a especificação do mesmo
- A implementação é apresentada à parte, no corpo do pacote

PACKAGES

- Sintaxe

```
CREATE OR REPLACE PACKAGE <nome> AS
<especificação de procedimento> |
< especificação de procedimento função>|
<declaração de variável> |
<definição de tipo> |
<declaração de exceção> |
<declaração de cursor>
END nome;
```


PACKAGES

- Exemplo 18: Pacote para cadastro de Veículos

```
CREATE OR REPLACE PACKAGE CadastroPackage
AS
 -- Insere veículo em 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);
```


PACKAGES

- Exemplo 18 (Cont.)

```
-- Remove um dado veículo de Carro
PROCEDURE RemoveCarro(p_chassi  IN
carro.chassi%TYPE);
-- Excecao levantada por RemoveCarro
e_carroNaoExistente EXCEPTION;
/* Tipo de tabela virtual utilizado para armazenar chassis */
TYPE t_chassiTable IS TABLE OF carro.chassi%TYPE
INDEX BY BINARY_INTEGER;
```


PACKAGES

- Exemplo 18 (Cont.)

```
/* Retorna uma tabela virtual contendo os chassis
   dos carros de um dado modelo */
PROCEDURE ListaChassi(p_modelo  IN
carro.modelo%TYPE, p_chassi OUT t_chassiTable,
p_Numcarros IN OUT BINARY_INTEGER);
END CadastroPackage;
/
```

Índice da tabela em memória

PACKAGES

- Exemplo 18: O corpo do pacote

Comandos para implementar as ações de cada procedure, function, etc.

```
CREATE OR REPLACE PACKAGE BODY
CadastroPackage AS
 -- Insere veículo em 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) IS
```

PACKAGES

- Exemplo 18 (Cont.)

```
INSERT INTO carro (chassi, modelo, km_carro,
data_carro)
   VALUES (p_chassi, p_modelo, p_km_carro,
p_data_carro);
COMMIT;
END InsereCarro;
-- Remove um dado veículo de Carro
PROCEDURE RemoveCarro(p_chassi  IN
carro.chassi%TYPE) IS
 BEGIN
   DELETE FROM carro
     WHERE chassi = p_chassi;
```


PACKAGES

- Exemplo 18 (Cont.)

```
/* Verificar se a operação DELETE teve sucesso. Se não
       existir a tupla, levantar erro. */
  IF SQL%NOTFOUND THEN
    RAISE e_carroNaoExistente;
  END IF;
  COMMIT;
END RemoveCarro;
```


PACKAGES

- Exemplo 18 (Cont.)

```
/* Retorna tabela PL/SQL contendo os
   chassis */
 PROCEDURE ListaChassi(p_modelo  IN
carro.modelo%TYPE, p_chassi OUT t_chassiTable,
p_Numcarros IN OUT BINARY_INTEGER) IS
v_chassi  carro.chassi%TYPE;
/* Cursor local para trazer registros
   de carros */
    CURSOR c_carros IS
      SELECT chassi
        FROM carro
        WHERE modelo = p_modelo;
```


PACKAGES

- Exemplo 18 (Cont.)

```
BEGIN
   /* p_Numcarros será o índice da tabela, que começará
em 0 e será incrementado pelo laço. No final ele terá o
número de tuplas trazidas e, portanto, o número de
linhas retornadas em p_chassi.*/

        p_Numcarros := 0;

  OPEN c_carros;
```


PACKAGES

- Exemplo 18 (Cont.)

```
LOOP
    FETCH c_carros INTO v_chassi;
    EXIT WHEN c_carros%NOTFOUND;
    p_Numcarros := p_Numcarros + 1;
    p_chassi(p_Numcarros) := v_chassi;
  END LOOP;
 END ListaChassi;
END CadastroPackage;
/
```

Todos os subprogramas definidos no Package precisam ter a implementação definida no Body

PACKAGES

- Cada elemento no pacote está no escopo deste e é visível fora dele por meio de sua qualificação
  - CadastroPackage.RemoveCarro
- É possível dar sobrecarga (overload) nos subprogramas de um pacote
  - Permite dar o mesmo nome a subprogramas distintos de um pacote definindo parâmetros diferentes em cada um deles

PACKAGES

- Inicialização de pacotes
  - Muitas vezes um pacote precisa ter um código de inicialização para a primeira vez que for executado
    - Deve ser fornecido no corpo do pacote
    - É uma chamada a um dos subprogramas do pacote em um bloco (BEGIN.. END) colocado no final do corpo

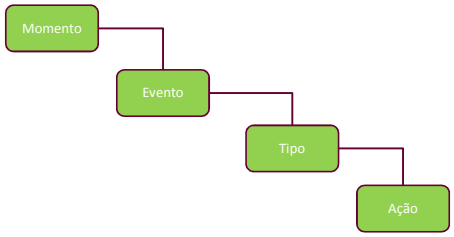
Triggers

Triggers (Gatilhos)

- Procedimentos criados e armazenados no SGBD que são automaticamente ativados em resposta a determinadas mudanças que ocorrem no BD
  - Podem ser Simple, Autonomous ou Compound
  - Em geral não são modificados e executam várias vezes
  - São executados implicitamente sempre que ocorre o evento que os dispara
    - O evento pode estar associado a uma tabela, uma view, um esquema ou ao banco de dados
  - Semelhantes a procedimentos, mas não podem ser locais em relação a um bloco, procedure, function ou package
  - Não aceitam parâmetros

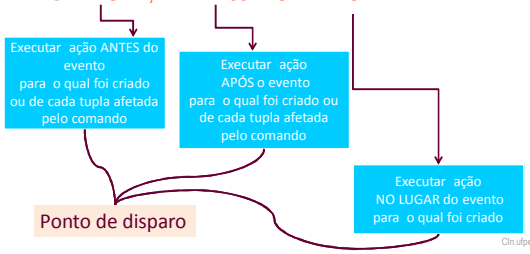
Triggers

- Um trigger é composto por quatro partes



Triggers - Momento

- Corresponde ao tempo em que o trigger deve ser executado
- Triggers do tipo simple são disparados exatamente em um dado ponto
  - BEFORE, AFTER ou INSTEAD OF



Triggers - Tipo

- Só aplicável a eventos de DML
  - Linha
    - Acionado para cada linha afetada
    - Para UPDATE de atributo requer a definição do atributo após OF
      - UPDATE OF <atributo> ON <tabela>
  - Comando
    - Trigger acionado independentemente do comando atualizar ou não uma ou mais linhas
    - Não permite acesso às linhas atualizadas

Triggers - Eventos

- Comando de manipulação no Banco de Dados (DML)
  - No modelo relacional: operações INSERT, DELETE, UPDATE
  - No orientado-a-objetos também na chamada de métodos
- Comando de definição no banco de dados
  - CREATE, ALTER, DROP
- Comando de operação do Banco de Dados
  - Ações de DBA: SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN, GRANT, REVOKE
- Eventos temporais, eventos externos
- Combinações dos eventos acima

Triggers - Ações

- Bloco PL/SQL executado em razão do evento
  - Requer a utilização de predicados lógicos quando o trigger pode ser disparado por mais de um evento DML

Predicado lógico	Significado
INSERTING	Quando o trigger for disparado por um INSERT
UPDATING	Quando o trigger for disparado por um UPDATE
DELETING	Quando o trigger for disparado por um DELETE

Triggers - Ações

- Especificação de ações
  - Pode ser uma sequência de comandos de modificação e acesso aos dados
  - Pode ser implícita, ou seja, a transação é abortada
  - Pode indicar um rollback da transação
  - Pode substituir a operação que causou o evento, por meio da palavra-chave instead (no caso do Oracle só para views)

Triggers

- Podem ser utilizados para
  - Manter restrições de integridade complexas
  - Efetuar auditoria às informações de uma tabela, registrando as alterações efetuadas e quem as efetuou (log seletivo)
  - Indicar automaticamente a outros programas que é necessário efetuar uma ação, quando são efetuadas alterações em uma tabela ou BD
  - Gerar valor de coluna (atributo)
  - Garantir regras de negócio
  - Controle de versões
  - Garantir restrições de acesso

Triggers - Recomendações

- Use gatilhos preferencialmente para
  - Garantir que quando uma operação for processada, ações relacionadas serão executadas
  - Impor regras de negócio complexas, impossíveis de definir usando restrições de integridade
  - Impor a integridade referencial quando tabelas pai e filho estão em diferentes nós de um banco de dados distribuído
  - Manter replicação síncrona de tabelas

Triggers - Recomendações

- Use gatilhos preferencialmente para (Cont.)
  - Apenas operações centralizadas, globais, que devem ser disparadas pelo comando que aciona o trigger, independentemente de qual usuário ou aplicação do BD tenha executado o referido comando
  - Modificar os dados de tabela quando comandos DML são emitidos sobre Views
- Use gatilhos ponderadamente
- Limite o tamanho de gatilhos (≈ 60 linhas)
  - Usar procedure, caso necessite maior tamanho

Triggers - Recomendações

- Não use gatilhos para
  - Refazer ações já existentes no SGBD, particularmente as que podem ser realizadas por
    - NOT NULL, UNIQUE
    - PRIMARY KEY
    - FOREIGN KEY
    - CHECK
    - DELETE CASCADE
    - DELETE SET NULL
- Não crie gatilhos recursivos

Cln.ufpe.br 97

Ativação de Triggers

- Ativação ocorre quando comandos são executados sobre uma tabela (DML) ou BD (DBA)
- Uso de BEFORE possibilita o acesso (consulta e alteração) a valores antigos (já existentes no BD - OLD) e novos (recém-inseridos - NEW)
  - O uso de AFTER só permite consulta
- Ativando gatilhos UMA ou VÁRIAS vezes
  - A opção FOR EACH ROW
    - Determina se o gatilho é do tipo *row trigger* (linha) ou *statement trigger (comando)*
      - Se especificada, o gatilho é executado UMA vez para cada tupla afetada pelo evento
      - Se omitida, o gatilho é executado UMA ÚNICA vez para cada ocorrência do evento

Cln.ufpe.br 98

Ativação Condicional de Triggers

- A opção WHEN
  - É usada apenas com *row triggers*
  - Consiste em uma expressão booleana – SQL
  - É avaliada para cada *tupla* afetada pelo evento
  - Apenas se o resultado da sua avaliação for verdadeiro, a ação é executada
  - Não tem efeito sobre a execução do evento que o dispara
  - Não pode incluir
    - Subconsultas
    - Expressões em PL/SQL
    - Funções definidas pelo usuário

Cln.ufpe.br 99

Triggers

- Sintaxe

Só para views

```
CREATE [OR REPLACE] TRIGGER <nome>
 [BEFORE | AFTER | INSTEAD OF ]  <evento>
ON  <tabela>
[REFERENCING  NEW   AS   <novo_nome>
              OLD   AS   <antigo_nome> ]
[FOR EACH ROW [ WHEN (<condição> )]]
[DECLARE   [PRAGMA
          AUTONOMOUS_TRANSACTION ]
[Definição de variáveis locais]]
BEGIN <corpo-do-procedimento>
END <nome>;
/
```

Evitar conflito com new ou old

Trigger de linha

Para ativação condicional

Definição de variáveis para uso no Trigger

Permitir controle de transação

Cln.ufpe.br 100

Triggers

- Restrições para trigger do tipo Simple
  - No Oracle, não podem emitir instruções de controle de transação
    - COMMIT, ROLLBACK ou SAVEPOINT
  - Não podem chamar nenhuma função que faça isso
  - Não podem declarar variável LONG ou LONG RAW

Permitidos para triggers do tipo Autonomous

Cln.ufpe.br 101

Triggers - Exemplos

- Trigger de linha
  - Exemplo 19: Na devolução do carro, alterar a quilometragem do carro em função da quilometragem final

```
CREATE OR REPLACE TRIGGER altera_km
AFTER UPDATE ON locacao
FOR EACH ROW
BEGIN
  IF :NEW.data_entrega IS NOT NULL THEN
     UPDATE carro SET km_carro = :NEW.km_final
          WHERE chassi = :NEW.chassi;
 END IF;
END;
/
```

Variável contendo valores inseridos

Usar :OLD - quando precisar acessar valores anteriores

Cln.ufpe.br 102

Triggers - Exemplos

- Trigger de linha - Impedir inserção de tupla inválida
  - Exemplo 20: Não permitir locação com carro tendo mais de três anos de uso

```
CREATE OR REPLACE TRIGGER verifica_data
BEFORE INSERT OR UPDATE ON locacao
FOR EACH ROW
DECLARE
  x date;
BEGIN
  SELECT data_carro INTO x FROM Carro WHERE chassi =
:NEW.chassi_carro;
IF sysdate - x >= 1059 THEN
  RAISE_APPLICATION_ERROR(-20011, 'Carro com mais de três anos de
uso');
END IF;
END;
/
```

Criar código para Mensagem de erro

Triggers - Exemplos

- Trigger de comando
  - Exemplo 21: Por razões contábeis da empresa, impedir a remoção de carro do BD no último dia de cada mês (considerar 28 como o último dia de fevereiro)

Definição de variáveis auxiliares

```
CREATE OR REPLACE TRIGGER dia_permitido
BEFORE DELETE ON Carro
DECLARE
 aux varchar2(2);
 x varchar2(2);
 y varchar2(2);
 ultimo_dia EXCEPTION;
```


Triggers - Exemplos

- Exemplo 21 (Cont.)

Extrai o mês de uma data

Data do sistema

```
BEGIN
  /* verificar último dia: */
  X := EXTRACT(month from sysdate);
  IF x IN ('1', '3', '5', '7', '8', '10', '12')
  THEN aux := '1'; END IF;
  IF x IN ('4', '6', '9', '11') THEN
    aux := '2'; END IF;
  IFx = 2 THEN aux := '3';
  END IF;
y := EXTRACT(day from sysdate);
```

Determinar número de dias que o mês contém

Extrai o dia de uma data

Triggers - Exemplos

- Exemplo 21 (Cont.)

Verifica se o dia é o último do mês

```
CASE aux
  WHEN 1 THEN IF y = '31' THEN RAISE ultimo_dia;
    END IF;
  WHEN 2 THEN IF y = '30' THEN RAISE ultimo_dia;
    END IF;
  WHEN 3 THEN IF y = '28'
    THEN RAISE ultimo_dia; END IF;
END CASE;
```


Triggers - Exemplos

- Exemplo 21 (Cont.)

Trata a exceção quando for o último dia do mês

```
EXCEPTION
 WHEN ultimo_dia THEN
  Raise_application_error(-20324,'ÚLTIMO DIA DO MÊS - '
   || 'Não é permitido remover carro do BD');
END dia_permitido;
/
```


Triggers

- Tabela mutante
  - Em alguns triggers é necessário fazer referência a valores da tabela que está sendo alterada
    - Em geral, quando um trigger tenta consultar a própria tabela que está sofrendo a ação
  - Também pode acontecer quando consultar uma tabela pai em um update/delete cascading

Um Trigger está tentando modificar ou consultar um dado que ainda está sendo modificado

Triggers

- Exemplo 22: Auditar inserções e modificações feitas na tabela Carro e gravar a ação realizada, o id da tupla acessada, o instante no qual ocorreu a ação e o número total de registros da tabela Carro em outra tabela (Carro_audit)

Tabela a ser auditada

Carro

Chassi	Modelo	Km_carro	Data_Carro
1234567	Sentra	100	15/04/2013

Triggers

- Exemplo 22 (Cont.)
  - Tabela de Auditoria

```
CREATE TABLE Carro_audit (
  id              NUMBER(10) NOT NULL,
  acao            VARCHAR2(10) NOT NULL,
  carro_id        NUMBER(7),
  numero_registros  NUMBER(10),
  hora_criacao  TIMESTAMP,
CONSTRAINT carro_audit_pk PRIMARY KEY (id)
);
CREATE SEQUENCE carro_audit_seq;
```

Quando ocorreu o evento
Identificação da linha da tabela
Ação realizada
Linha acessada da tabela
Total de linhas da tabela

Triggers

- Exemplo 22(Cont.)
  - As ações do Trigger para fazer a auditoria devem ser definidas em um Package

```
CREATE OR REPLACE PACKAGE trigger_api AS
PROCEDURE carro_row_change (p_id IN
    carro.chassi%TYPE, p_acao IN  VARCHAR2);

END trigger_api;
/
```


Triggers

- Exemplo 22 (Cont.) – Corpo do pacote

```
CREATE OR REPLACE PACKAGE BODY trigger_api AS
PROCEDURE carro_row_change (p_id      IN  carro.chassi%TYPE,
                p_acao  IN  VARCHAR2) IS
 l_count  NUMBER(10);
BEGIN
 SELECT COUNT(*)
 INTO   l_count
 FROM   Carro;
```


Triggers

- Exemplo 22 (Cont.)

```
INSERT INTO carro_audit (id, acao, carro_id, numero_registros,
                hora_criacao)
  VALUES (carro_audit_seq.NEXTVAL, p_acao, p_id,
    l_count, SYSTIMESTAMP);
END carro_row_change;
END trigger_api;
/
```


Triggers

- Exemplo 22 (Cont.)
  - Trigger por linha para que cada inserção ou modificação na tabela Carro seja auditada na tabela carro_audit

```
CREATE OR REPLACE TRIGGER carro_trg
AFTER INSERT OR UPDATE ON carro
FOR EACH ROW
BEGIN
  IF inserting THEN
    trigger_api.carro_row_change(p_id => :new.chassi,
            p_acao => 'INSERT');
  ELSE
    trigger_api.carro_row_change(p_id => :new.chassi,
      p_acao => 'UPDATE');  END IF;   END;
/
```

Triggers

- Exemplo 22 – Após criação do Trigger (Cont.)
  - Inserir um registro na tabela Carro

```
INSERT INTO carro (Chassi, Modelo, Km_carro, Data_Carro)
VALUES (2314567, 'HB20', 35,to_date('12/10/2013',
 'dd/mm/yyyy'));
```

```
ERROR at line 1:
ORA-04091: table U_FDFD.CARRO is mutating, trigger/function may not see it
ORA-06512: at "U_FDFD.TRIGGER_API", line 6
ORA-06512: at "U_FDFD.CARRO_TRG", line 3
ORA-04088: error during execution of trigger 'U_FDFD.CARRO_TRG'
```


Triggers

- Possíveis soluções para erros de tabelas mutantes
  - Uso combinado de trigger por linha e outro por comando
  - A partir da versão 11g release 1, a Oracle implementou o conceito de ***Compound Triggers***
  - Uso de ***autonomous transactions***

Tabela Mutante: Solução usando combinação de trigger de linha (row) e trigger de comando (statement)

Triggers

- Solução com o uso combinado de trigger de linha (row) e outro de comando (statement) com as ações implementadas em um package
  - Exemplo 23: Resolver o problema da tabela mutante do Exemplo 22

Substitui o package existente

Para trigger de linha

```
CREATE OR REPLACE PACKAGE trigger_api AS
PROCEDURE carro_row_change (p_id     IN  carro.chassi%TYPE,
                            p_acao  IN  VARCHAR2);
PROCEDURE carro_statement_change;
END trigger_api;
```

Para trigger de comando

Triggers

- Exemplo 23 (Cont.)

```
CREATE OR REPLACE PACKAGE BODY trigger_api  AS
TYPE t_change_rec IS RECORD (
  id      carro.chassi%TYPE,
  acao  carro_audit.acao%TYPE
);
TYPE t_change_tab IS TABLE OF t_change_rec;
g_change_tab  t_change_tab := t_change_tab();
```

Criando estrutura de tabela na memória

Triggers

- Exemplo 23 (Cont.)

```
PROCEDURE carro_row_change (p_id     IN  carro.chassi%TYPE,
                            p_acao  IN  VARCHAR2) IS
BEGIN
  g_change_tab.extend;
  g_change_tab(g_change_tab.last).id     := p_id;
  g_change_tab(g_change_tab.last).acao := p_acao;
END carro_row_change;
```

Instanciando a tabela em memória

Triggers

- Exemplo 23 (Cont.)

Consultando a tabela em memória

```
PROCEDURE carro_statement_change IS
  l_count  NUMBER(10);
BEGIN
  FOR i IN g_change_tab.first .. g_change_tab.last LOOP
    SELECT COUNT(*)
    INTO   l_count
    FROM   carro;
    INSERT INTO carro_audit (id, acao, carro_id,
           numero_registros, hora_criacao)
    VALUES (carro_audit_seq.NEXTVAL,
        g_change_tab(i).acao, g_change_tab(i).id,
l_count, SYSTIMESTAMP);
```

Inserindo dados na tabela de Auditoria

Triggers

- Exemplo 23 (Cont.)

Removendo tabela da memória

```
END LOOP;
  g_change_tab.delete;
END carro_statement_change;
END trigger_api;
```


Triggers

- Exemplo 23 (Cont.)
  - Novo trigger

```
CREATE OR REPLACE TRIGGER carro_st_trg
AFTER INSERT OR UPDATE ON carro
BEGIN
  trigger_api.carro_statement_change;
END;
```

O outro trigger (**carro_trg**) permanece, com as ações alteradas no package

Triggers

- Exemplo 23 (Cont.)
  - Testando, inserindo novo carro na tabela

```
INSERT INTO carro (Chassi, Modelo, Km_carro, Data_Carro)
VALUES ('2314567', 'HB20', 35,to_date('12/10/2013',
 'dd/mm/yyyy'));
```

1 row created.

Carro

Chassi	Modelo	Km_carro	Data_Carro
1234567	Sentra	100	15-APR-13
2314567	HB20	35	12-OCT-13

Carro_audit

ID	ACAO	CARRO_ID	NUMERO_ REGISTROS	HORA_CRIACAO
2	INSERT	2314567	2	06-MAY-14 02.00.59.739123 PM

Triggers

- Exemplo 23 (Cont.) – Testando inserindo novo carro

```
INSERT INTO carro (Chassi, Modelo, Km_carro, Data_Carro)
VALUES ('4013568', 'UP', 15,to_date('20/02/2014',
 'dd/mm/yyyy'));
```

1 row created.

Carro

Chassi	Modelo	Km_carro	Data_Carro
1234567	Sentra	100	15-APR-13
2314567	HB20	35	12-OCT-13
4013568	UP	15	20-FEB-14

Triggers

- Exemplo 23 (Cont.)

Carro_audit

ID	ACAO	CARRO_ID	NUMERO_ REGISTROS	HORA_CRIACAO
2	INSERT	2314567	2	06-MAY-14 02.00.59.739123 PM
3	INSERT	4013568	3	06-MAY-14 02.30.56.020284 PM

Triggers

- Exemplo 23 (Cont.) – Testando com UPDATE

```
UPDATE carro SET modelo = modelo;
3 rows updated.
```

Carro

Chassi	Modelo	Km_carro	Data_Carro
1234567	Sentra	100	15-APR-13
2314567	HB20	35	12-OCT-13
4013568	UP	15	20-FEB-14

Carro_audit

ID	ACAO	CARRO_ID	NUMERO_ REGISTROS	HORA_CRIACAO
2	INSERT	2314567	2	06-MAY-14 02.00.59.739123 PM
3	INSERT	4013568	3	06-MAY-14 02.30.56.020284 PM
4	UPDATE	1234567	3	06-MAY-14 02.42.46.191719 PM
5	UPDATE	2314567	3	06-MAY-14 02.42.46.191858 PM
6	UPDATE	4013568	3	06-MAY-14 02.42.46.191961 PM

Tabela Mutante: Solução utilizando Compound Trigger

Compound Triggers

- Podem ser disparados a partir de mais de um ponto
- Tornam mais fácil programar uma abordagem onde se quer executar ações para vários pontos de disparo, compartilhando dados comuns
- Organizar as ações do trigger por diferentes pontos de disparo

Compound Triggers

- Sintaxe

```
CREATE [OR REPLACE] TRIGGER <nome>
FOR  <evento>  ON  <tabela>
COMPOUND TRIGGER
<Declaração de Variáveis Locais>
BEFORE | AFTER STATEMENT IS
BEGIN
<bloco>
END BEFORE | AFTER STATEMENT;
....
BEFORE | AFTER EACH ROW IS
BEGIN
<bloco>
END BEFORE | AFTER EACH ROW;
....
END <nome> ;
```

Ações para antes ou depois da execução do comando que dispara o trigger

Ações para antes ou depois da execução do comando em cada linha da tabela considerada

Compound Triggers

- Exemplo 24: Solução para o problema da auditoria em Carro, considerando também a remoção de carro

```
CREATE OR REPLACE TRIGGER audit_carro
 FOR INSERT OR UPDATE  OR DELETE ON Carro
COMPOUND TRIGGER
x INTEGER;
cadeia VARCHAR2(6);
BEFORE STATEMENT IS
BEGIN
SELECT COUNT(*) INTO x from carro;
END BEFORE STATEMENT;
```

Ações para ponto de disparo antes da execução do comando

Compound Triggers

- Exemplo 24 (Cont.)

```
AFTER EACH ROW IS
BEGIN
IF INSERTING THEN cadeia := 'INSERT';
      ELSEIF DELETING THEN cadeia := 'DELETE';
ELSE cadeia := 'UPDATE';
END IF;
INSERT INTO carro_audit (id, acao, carro_id, numero_registros,
                hora_criacao)
  VALUES (carro_audit_seq.NEXTVAL, cadeia, :NEW.id,
            x, SYSTIMESTAMP);
END AFTER EACH ROW;
END audit_carro;
```

Ações para após atualização de cada tupla

Compound Triggers

- Exemplo 24 (Cont.)
  - Testando, inserindo mais um carro

```
INSERT INTO carro (Chassi, Modelo, Km_carro, Data_Carro)
VALUES ('5321456', 'Cruze', 12,to_date('10/04/2014',
'dd/mm/yyyy'));
```

1 row created.

Carro

Chassi	Modelo	Km_carro	Data_Carro
1234567	Sentra	100	15-APR-13
2314567	HB20	35	12-OCT-13
4013568	UP	15	20-FEB-14
5321456	Cruze	12	10-APR-14

Compound Triggers

- Exemplo 24 (Cont.)

Carro_audit

ID	ACAO	CARRO_ID	NUMERO_ REGISTROS	HORA_CRIACAO
2	INSERT	2314567	2	06-MAY-14 02.00.59.739123 PM
3	INSERT	4013568	3	06-MAY-14 02.30.56.020284 PM
4	UPDATE	1234567	3	06-MAY-14 02.42.46.191719 PM
5	UPDATE	2314567	3	06-MAY-14 02.42.46.191858 PM
6	UPDATE	4013568	3	06-MAY-14 02.42.46.191961 PM
7	INSERT	5321456	4	06-MAY-14 02.59.07.033810 PM

Compound Triggers

- Exemplo 24 (Cont.)
  - Testando, removendo a tupla do carro com chassi igual a 2314567

```
DELETE FROM carro WHERE chassi = 2314567;
```

1 row deleted.

Carro

Chassi	Modelo	Km_carro	Data_Carro
1234567	Sentra	100	15-APR-13
4013568	UP	15	20-FEB-14
5321456	Cruze	12	10-APR-14

O carro HB20 foi removido

Compound Triggers

- Exemplo 24 (Cont.)

Carro_audit

ID	ACAO	CARRO_ID	NUMERO_ REGISTROS	HORA_CRIACAO
2	INSERT	2314567	2	06-MAY-14 02.00.59.739123 PM
3	INSERT	4013568	3	06-MAY-14 02.30.56.020284 PM
4	UPDATE	1234567	3	06-MAY-14 02.42.46.191719 PM
5	UPDATE	2314567	3	06-MAY-14 02.42.46.191858 PM
6	UPDATE	4013568	3	06-MAY-14 02.42.46.191961 PM
7	INSERT	5321456	4	06-MAY-14 02.59.07.033810 PM
8	DELETE	2314567	3	06-MAY-14 02.59.07.034012 PM

Tabela Mutante: Solução utilizando *autonomous transactions*

Trigger com Autonomuos Transaction

- Permite executar uma nova transação, independente da transação que dispara o trigger
  - Na seção DECLARE do trigger utilizar
    - PRAGMA AUTONOMOUS_TRANSACTION;
  - Como são duas transações, sem controle pode levar a deadlock, provocando rollback da transação que dispara o trigger
    - A transação do trigger deve sofrer commit
  - A tabela vista pela transação autonôma é a com os dados anteriores à ação da transação que disparou o trigger

Trigger com Autonomuos Transaction

- Sintaxe

Transação autônoma para o trigger

Efetivar a transação do trigger

```
CREATE [OR REPLACE] TRIGGER <nome>
FOR  <evento>  ON  <tabela>
FOR EACH ROW
 DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
...
BEGIN
<bloco>
COMMIT;
END <nome> ;
```


Trigger com Autonomuos Transaction

- Exemplo 25: Solução para o problema da auditoria em Carro

```
CREATE OR REPLACE TRIGGER carro_trg
  AFTER INSERT OR UPDATE ON carro
  FOR EACH ROW
  DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
  l_count  INTEGER;
BEGIN
  SELECT COUNT(*)
  INTO   l_count
  FROM   Carro;
```


Trigger com Autonomuos Transaction

- Exemplo 25 (Cont.)

```
IF inserting THEN l_count:= l_count + 1;
  INSERT INTO carro_audit VALUES (carro_audit_seq.NEXTVAL,
   'INSERT', :new.chassi, l_count, SYSTIMESTAMP);
ELSE
  INSERT INTO carro_audit VALUES (carro_audit_seq.NEXTVAL,
   'UPDATE', :new.chassi, l_count, SYSTIMESTAMP);
END IF;
COMMIT;
END;
```


Trigger com Autonomuos Transaction

- Exemplo 25 (Cont.)

Tabela a ser auditada

Carro

Chassi	Modelo	Km_carro	Data_Carro
1234567	Sentra	100	15/04/2013

Trigger com Autonomuos Transaction

- Exemplo 25 (Cont.)
  - Testando, inserindo mais um carro

```
INSERT INTO carro (Chassi, Modelo, Km_carro, Data_Carro)
VALUES ('2314567', 'HB20', 35,to_date('12/10/2013',
'dd/mm/yyyy'));
```

CHASSI	MODELO	KM_CARRO	DATA_CARRO
2314567	HB20	35	12/10/13
1234567	Sentra	100	15/04/13

ID	ACAO	CARRO_ID	NUMERO_REGISTROS	HORA_CRIACAO
6	INSERT	2314567	2	22-OUT-14 03.51.21.413000 TARDE

Trigger com Autonomuos Transaction

- Exemplo 25 (Cont.)
  - Testando, inserindo mais um carro

```
INSERT INTO carro (Chassi, Modelo, Km_carro, Data_Carro)
VALUES ('4013568', 'UP', 15,to_date('20/02/2014',
'dd/mm/yyyy'));
```

CHASSI	MODELO	KM_CARRO	DATA_CARRO
2314567	HB20	35	12/10/13
4013568	UP	15	20/02/14
1234567	Sentra	100	15/04/13

ID	ACAO	CARRO_ID	NUMERO_REGISTROS	HORA_CRIACAO
6	INSERT	2314567	2	22-OUT-14 03.51.21.413000 TARDE
7	INSERT	4013568	3	22-OUT-14 03.56.26.403000 TARDE

Trigger com Autonomuos Transaction

- Exemplo 25 (Cont.)
  - Testando, atualizando modelo