Gerenciamento de Dados e Informação

Fernando Fonseca
Ana Carolina
Robson Fidalgo

Centro de Informática
Universidade Federal de Pernambuco
CIn.ufpe.br

PL/SQL

- Procedural Language/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

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

Variáveis

- Tipos
  - Mesmos usados pelo Oracle

Numérico	BINARY_INTEGER inteiro de -231 - 1 a 231 -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 apostrofo ' '. Para fazer operações usar funções do Oracle para Date-Time

Registros

- Declaração e uso

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

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

```
...
<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 análoga às Tabelas Relacionais
  - Só existe em memória principal, durante a execução do progrma PL
  - Acesso só pode ocorrer através da indexação dos elementos da tabela
    - Não se pode usar SELECT, INSERT, etc.
  - 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

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

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

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


Operadores

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

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;
```


Declaração e Inicialização de Variáveis

- Declaração de constante

```
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;
```


Estrutura de Controle

- Comando IF-THEN-ELSE

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


Estrutura 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;
```


Estrutura de Controle

- Comando CASE

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


Estrutura 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
    - Aumentar 2 no valor de uma variável C, enquanto A < B

```
WHILE A < B LOOP
C := C + 2;
END LOOP;
```


Laços

- Laço FOR

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

  - Exemplo
    - Aumentar 2 no valor de uma variável C, dez vezes

```
FOR i IN 1.. 10
 LOOP
C := C + 2;
END LOOP;
```


Laços

- GOTO e Rótulos

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

  - Laços podem ser etiquetados para uso em EXIT

Recuperação de Dados para Variável com SELECT

- Utilizar comando → SELECT ... INTO

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

- Considere
  - Número de <variável(is)> deve ser igual ao número de <atributo(s)>
  - Os tipos de cada <atributo> e da <variável> correspondente devem ser compatíveis
  - Deve ser recuperada uma única tupla

Recuperação de Dados para Variável com SELECT

- Considere (Cont.)
  - <variável(is)> devem ser declaradas

```
qtdEmp  NUMBER;
```

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

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

Saída de Dados

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

```
Set serveroutput on;
```

  - Comandos de saída
    - dbms_output.put(' ... ') ou → Escreve na mesma linha
    - dbms_output.put_line (' ... ') → Escreve e depois muda de linha

Saída de Dados

- Na interface de caracteres (SQL+) (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

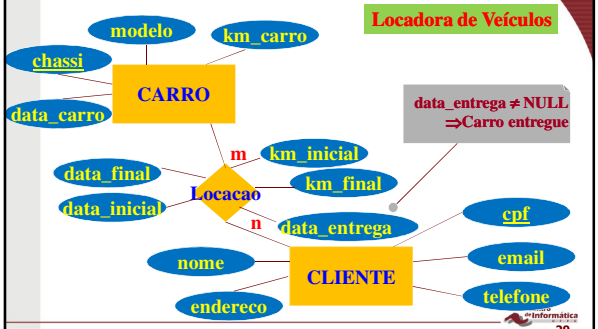
Saída de Dados

- Na interface de caracteres (SQL+) (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



Blocos

Blocos

- Várias instruções de SQL podem estar contidas em um único bloco de PL/SQL e 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

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

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

END;
```

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


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;
```


Blocos

- Exemplo (Cont.)

```
EXCEPTION
/* Verifica se o registro foi encontrado. Em caso contrário,
insere. */
  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;
```

37

Tratamento de Exceções

38

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 */
```

39

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;
```

40

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;
/
```

41

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

42

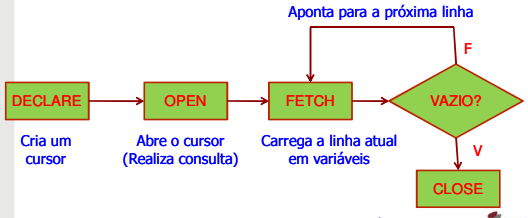
Cursores

Cursores

- Utilizados para processar várias linhas obtidas a partir da base de dados (com 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>;
FETCH <nome> INTO  <registro>;
```

  - Fechar o cursor

```
CLOSE <nome>;
```


Cursores

- 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

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';
```

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;
```


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*/
```


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;
/
```


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;
```


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;
/
```


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;
/
```

55

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;
```

56

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

57

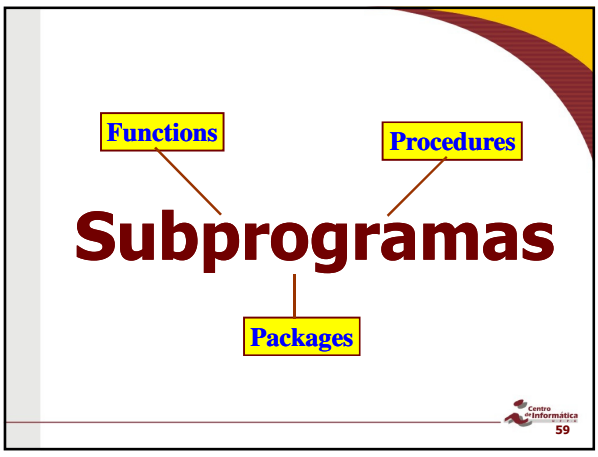
Cursores

- 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

58



59

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

60

Procedimentos

- Sintaxe

```
CREATE [OR REPLACE] PROCEDURE <nome>
 [(parâmetro [{IN | OUT | IN OUT}] tipo, ....)]
{IS | AS}
<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 [{IN | OUT | IN OUT}] tipo, ....)]
RETURN  <tipo-retorno> {IS | AS}
BEGIN <corpo-da-função>
END <nome>;
```

- 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

```
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;
```

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 através do corpo do programa

PACKAGES

- Sintaxe

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


PACKAGES

- Exemplo: 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 (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 utilizado para armazenar          chassis */
TYPE t_chassiTable IS TABLE OF carro.chassi%TYPE
INDEX BY BINARY_INTEGER;
```


PACKAGES

- Exemplo (Cont.)

```
/* Retorna uma tabela 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;
/
```


PACKAGES

- O corpo do pacote

```
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

- O corpo do pacote (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

- O corpo do pacote (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

- O corpo do pacote (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

- O corpo do pacote (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

- O corpo do pacote (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;
/
```


PACKAGES

- Cada elemento no pacote está no escopo deste e é visível fora dele através de sua qualificação
  - CadastroPackage.RemoveCarro
- É possível dar sobrecarga (overload) nos subprogramas de um pacote
  - Permite dar o mesmo nome a subprogramas 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

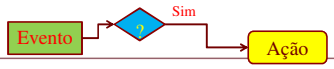
79

Triggers

80

Triggers (Gatilhos)

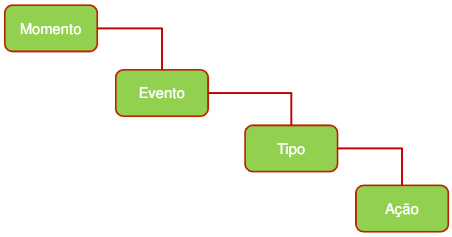
- Procedimentos armazenados no SGBD que são automaticamente ativados em resposta a determinadas mudanças que ocorrem no BD
  - Em geral não são modificados e executam várias vezes
  - São executados implicitamente sempre que ocorre o evento que os dispara
  - Semelhantes a procedimentos, mas não podem ser locais em relação a um bloco
  - Não aceitam parâmetros



81

Triggers (Gatilhos)

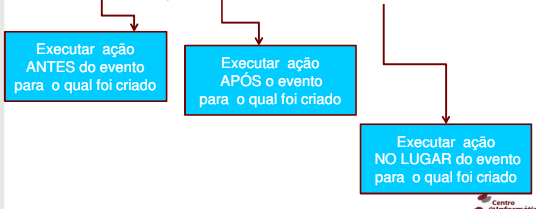
- Um trigger é composto por quatro partes



82

Triggers - Momento

- Corresponde ao tempo em que o trigger deve ser executado
  - BEFORE, AFTER ou INSTEAD OF



83

Triggers - Tipo

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

84

Triggers - Eventos

- Modificações no banco de dados
  - No relacional (DML): operações INSERT, DELETE, UPDATE
  - No orientado-a-objetos também na chamada de métodos
  - Operações de transação: COMMIT, ABORT, PREPARE-TO-COMMIT
  - Ações de DBA: CREATE, ALTER, DROP, SERVERERROR,LOGON,LOGOFF,STARTUP, SHUTDOWN, GRANT, REVOKE, ...
  - Eventos temporais, eventos externos
  - A composição dos eventos acima

Triggers - Ações

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

Predicado	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 seqüência 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, através 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 negócios
  - Controle de versões
  - Garantir restrições de acesso

Triggers - Recomendações

- Use gatilhos para garantir que quando uma operação for processada, ações relacionadas serão executadas
- Não defina gatilhos para refazer ações já existentes no SGBD
- Limite o tamanho de gatilhos (≈ 60 linhas)
  - Usar procedure
- Use gatilhos apenas para ações globais
- Não crie gatilhos recursivos
- Use gatilhos ponderadamente

Ativação de Triggers

- Ativação ocorre quando comandos são executados sobre uma tabela (DML) ou BD (DBA)
- Uso de BEFORE possibilita o acesso a valores antigos (já existentes no BD - OLD) e novos (recém-inseridos - NEW)
- 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 de evento

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
  - Não pode incluir
    - Subconsultas
    - Expressões em PL/SQL
    - Funções definidas pelo usuário

Triggers

- Sintaxe

```
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 ]]
BEGIN <corpo-do-procedimento>
END <nome>;
```


Triggers

- Cláusula INSTEAD OF
  - Só para views
- Cláusula REFERENCING
  - Evitar conflito com new ou old
- Cláusula FOR EACH ROW
  - Trigger de linha (row trigger)
- Cláusula WHEN
  - Colocar condição
- Cláusula PRAGMA_AUTONOMUS
  - Controle de transação

Triggers

- Restrições
  - 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

Exceto Trigger Autônomo

Triggers - Exemplo

- 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

:OLD -Variável contendo valores anteriores

Triggers - Exemplo

- Impedir inserção de valor inválido

```
CREATE OR REPLACE TRIGGER verifica_situacao
BEFORE INSERT OR UPDATE ON locacao
FOR EACH ROW
BEGIN
  IF :NEW.km_final < :NEW.km_inicial THEN
  RAISE_APPLICATION_ERROR(  -20011,  'Km_Final  <
Km_Inicial');
END  IF;
END;
/
```

Criar código para Mensagem de erro

Triggers

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

Um Trigger está tentando mudar ou examinar um dado que ainda está sendo mudado

97

Triggers

- Exemplo
  - Auditar as ações feitas em uma tabela (tab1) e gravar o número total de registros em outra tabela (tab1_audit)

```
CREATE TABLE tab1 (
  id           NUMBER(10) NOT NULL,
  description  VARCHAR2(50) NOT NULL,
  CONSTRAINT tab1_pk PRIMARY KEY (id)
);

CREATE SEQUENCE tab1_seq;
```

98

Triggers

- Inserir dados na tabela tab1

```
INSERT INTO tab1 (id, description)
VALUES (tab1_seq.NEXTVAL,
'Three');
```

ID	DESCRIPTION
1	Three

99

Triggers

- Exemplo (Cont.)

```
CREATE TABLE tab1_audit (
  id           NUMBER(10) NOT NULL,
  action       VARCHAR2(10) NOT NULL,
  tab1_id      NUMBER(10),
  record_count NUMBER(10),
  created_time TIMESTAMP,
CONSTRAINT tab1_audit_pk PRIMARY KEY (id),
CONSTRAINT tab1_audit_tab1_fk FOREIGN KEY (tab1_id)
  REFERENCES tab1(id)
);
CREATE SEQUENCE tab1_audit_seq;
```

ID	ACTION	TAB1_ID	RECORD_COUNT	CREATED_TIME

100

Triggers

- Exemplo (Cont.)
  - Código das ações do Trigger para fazer a auditoria em um Package

```
CREATE OR REPLACE PACKAGE trigger_api AS
PROCEDURE tab1_row_change (p_id IN
    tab1.id%TYPE, p_action IN  VARCHAR2);

END trigger_api;
/
```

101

Triggers

- Exemplo (Cont.)

```
CREATE OR REPLACE PACKAGE BODY trigger_api AS
PROCEDURE tab1_row_change (p_id      IN  tab1.id%TYPE,
                           p_action  IN  VARCHAR2) IS
 l_count  NUMBER(10);
BEGIN
 SELECT COUNT(*)
 INTO   l_count
 FROM   tab1;
…
```

102

Triggers

- Exemplo (Cont.)

```
INSERT INTO tab1_audit (id, action, tab1_id, record_count,
                        created_time)
  VALUES (tab1_audit_seq.NEXTVAL, p_action, p_id,
    l_count, SYSTIMESTAMP);
END tab1_row_change;
END trigger_api;
```


Triggers

- Exemplo (Cont.)
  - Trigger por linha para que cada mudança na tabela tab1 seja auditada na tab1_audit

```
CREATE OR REPLACE TRIGGER tab1_ariu_trg
AFTER INSERT OR UPDATE ON tab1
FOR EACH ROW
BEGIN
 IF inserting THEN
   trigger_api.tab1_row_change(p_id => :new.id,
                p_action => 'INSERT');
 ELSE
   trigger_api.tab1_row_change(p_id => :new.id,
    p_action => 'UPDATE');  END IF;   END;
```


Triggers

- Exemplo (Cont.)
  - Inserir um registro na tabela tab1

```
INSERT INTO tab1 (id, description)
VALUES (tab1_seq.NEXTVAL, 'ONE');
```

```
ORA-04091: a tabela FERNANDO.TAB1 é mutante;
talvez o gatilho/função não possa localizá-la
ORA-06512: em "FERNANDO.TRIGGER_API", line 6
ORA-06512: em "FERNANDO.TAB1_ARIU_TRG", line 3
ORA-04088: erro durante a execução do
gatilho 'FERNANDO.TAB1_ARIU_TRG'
```


Triggers

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

Triggers

- Uso combinado de trigger por linha e por statement com as ações implementadas por package

```
CREATE OR REPLACE PACKAGE trigger_api AS
PROCEDURE tab1_row_change (p_id      IN  tab1.id%TYPE,
                           p_action  IN  VARCHAR2);
PROCEDURE tab1_statement_change;
END trigger_api;
```


Triggers

- Exemplo (Cont.)

```
CREATE OR REPLACE PACKAGE BODY trigger_api  AS
TYPE t_change_rec IS RECORD (
  id      tab1.id%TYPE,
  action  tab1_audit.action%TYPE
);
TYPE t_change_tab IS TABLE OF t_change_rec;
g_change_tab  t_change_tab := t_change_tab();

...
```

Triggers

- Exemplo (Cont.)

```
...
PROCEDURE tab1_row_change (p_id      IN  tab1.id%TYPE,
                           p_action  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).action := p_action;
END tab1_row_change;
...
```


Triggers

- Exemplo (Cont.)

```
...
PROCEDURE tab1_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   tab1;
    INSERT INTO tab1_audit (id, action, tab1_id,
           record_count, created_time)
    VALUES (tab1_audit_seq.NEXTVAL,
        g_change_tab(i).action, g_change_tab(i).id,
l_count, SYSTIMESTAMP);
...
```


Triggers

- Exemplo (Cont.)

```
...
END LOOP;
  g_change_tab.delete;
END tab1_statement_change;
END trigger_api;
```


Triggers

- Exemplo (Cont.)
  - Novo trigger

```
CREATE OR REPLACE TRIGGER TAB1_ASIU_TRG
AFTER INSERT OR UPDATE ON tab1
BEGIN
  trigger_api.tab1_statement_change;
END;
```


Triggers

- Exemplo (Cont.)
  - Testando

```
INSERT INTO tab1 (id, description)
    VALUES (tab1_seq.NEXTVAL, 'ONE');

1 linha criada.
```

ID	DESCRIPTION
1	Three
3	ONE

ID	ACTION	TAB1_ID	RECORD_COUNT	CREATED_TIME
1	INSERT	3	2	29-MAR-09 08.30.37,150000 TARDE

Triggers

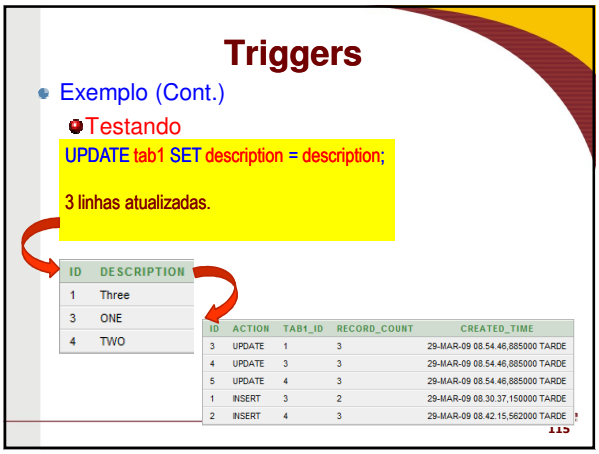
- Exemplo (Cont.)
  - Testando

```
INSERT INTO tab1 (id, description)
    VALUES (tab1_seq.NEXTVAL, 'TWO');

1 linha criada.
```

ID	DESCRIPTION
1	Three
3	ONE
4	TWO

ID	ACTION	TAB1_ID	RECORD_COUNT	CREATED_TIME
1	INSERT	3	2	29-MAR-09 08.30.37,150000 TARDE
2	INSERT	4	3	29-MAR-09 08.42.15,562000 TARDE


Triggers
Exemplo (Cont.)
Testando
UPDATE tab1 SET description = description;
3 linhas atualizadas.
ID DESCRIPTION
1 Three
3 ONE
4 TWO
ID ACTION TAB1_ID RECORD_COUNT CREATED_TIME
3 UPDATE 1 3 29-MAR-09 08.54.46,885000 TARDE
4 UPDATE 3 3 29-MAR-09 08.54.46,885000 TARDE
5 UPDATE 4 3 29-MAR-09 08.54.46,885000 TARDE
1 INSERT 3 2 29-MAR-09 08.30.37,150000 TARDE
2 INSERT 4 3 29-MAR-09 08.42.15,562000 TARDE