SQL – PSM
(Persistent Stored Modules)

UFCG/CEEI/DSC
Banco de Dados I
Prof. Cláudio Baptista, Ph.D.

Motivação

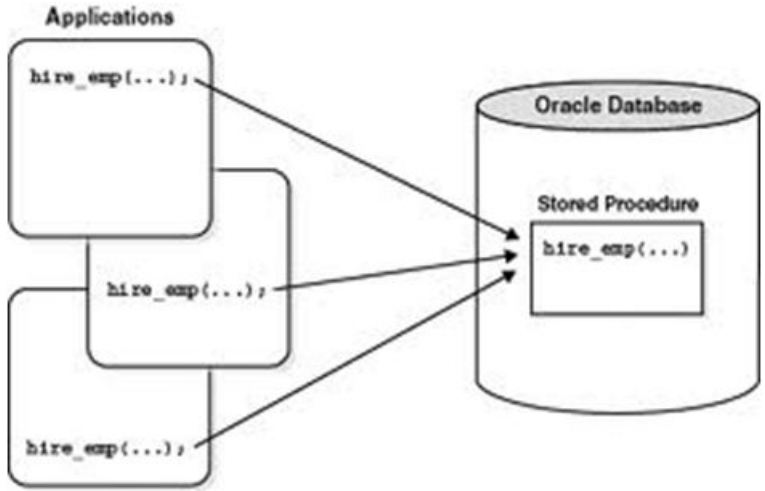
- SQL é bom, mas...
  - Como verificar condições? Como iterar?
  - Como verificar erros durante a manipulação de dados?
  - Instruções SQL são passadas para o SGBD uma por vez;
    - Fazer requisições de consultas complexas envolvendo junções com muitas tabelas torna-se custoso.

SQL – PSM

- SQL – PSM (Persistent Storage Module): padrão SQL para elaboração de Procedures e Functions que são armazenadas e executam no SGBD
- Cada SGBD oferece sua própria linguagem (PL/SQL, Transact/SQL, PL/pgSQL, etc.);
- Em PSM, definimos módulos que são coleções de definições de funções ou procedimentos, declarações de tabelas temporárias, dentre outros.

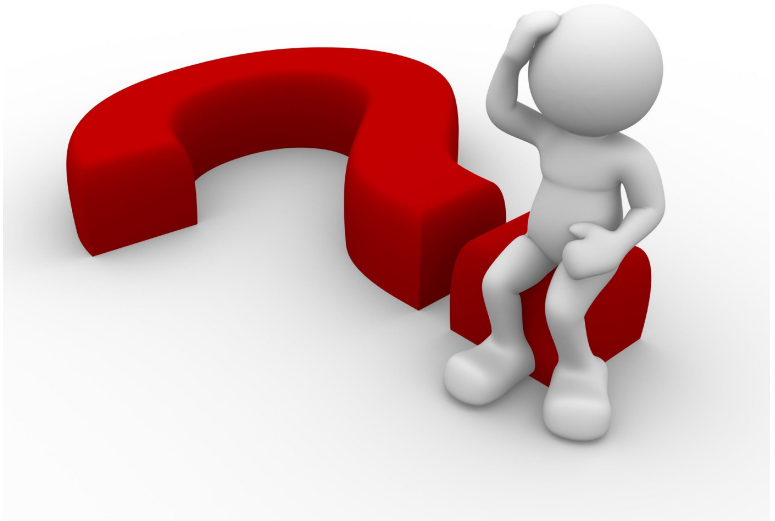
Vantagens

- Aplicações eficientes para grandes volumes de dados;
  - Processamento é feito do lado do servidor;
- Não necessita de API's intermediárias (JDBC, por exemplo);
- Custo de manutenção relativamente baixo;



Desvantagens

- Código fica preso ao SGBD
- Fica preso às limitações da implementação
- Difícil de portar as rotinas entre SGBDs distintos



PL/SQL

- Linguagem de desenvolvimento do SGBD Oracle que “implementa” o padrão SQL/PSM (não é completamente fiel ao padrão)

- Significa “Procedural Language extensions to SQL”

- Permite variáveis locais, laços, condições, procedures, consulta à relações “one tuple at a time”, etc.

- Forma geral:

```
DECLARE
    declarações (optativo)
BEGIN
    comandos executáveis; (obrigatórios)
    EXCEPTION
            comandos para manipular erros (optativo)
END;
```

PL/SQL

- Código PL/SQL é feito de blocos com uma única estrutura

- Existem dois tipos de blocos em PL/SQL:
  a. Blocos Anônimos: não possuem nomes (são como scripts)
     - Podem ser escritos e executados imediatamente;
     - Podem ser usados em um trigger (gatilho), para implementar a Ação.
  b. Blocos Nomeados: são armazenados no banco de dados
     - Procedures;
     - Functions;
     - Pacotes.

Blocos Anônimos

DECLARE (opcional)

/* aqui se declaram as variáveis que serão usadas no bloco */

BEGIN (obrigatório)

/* define-se os comandos que dizem o que o bloco faz*/

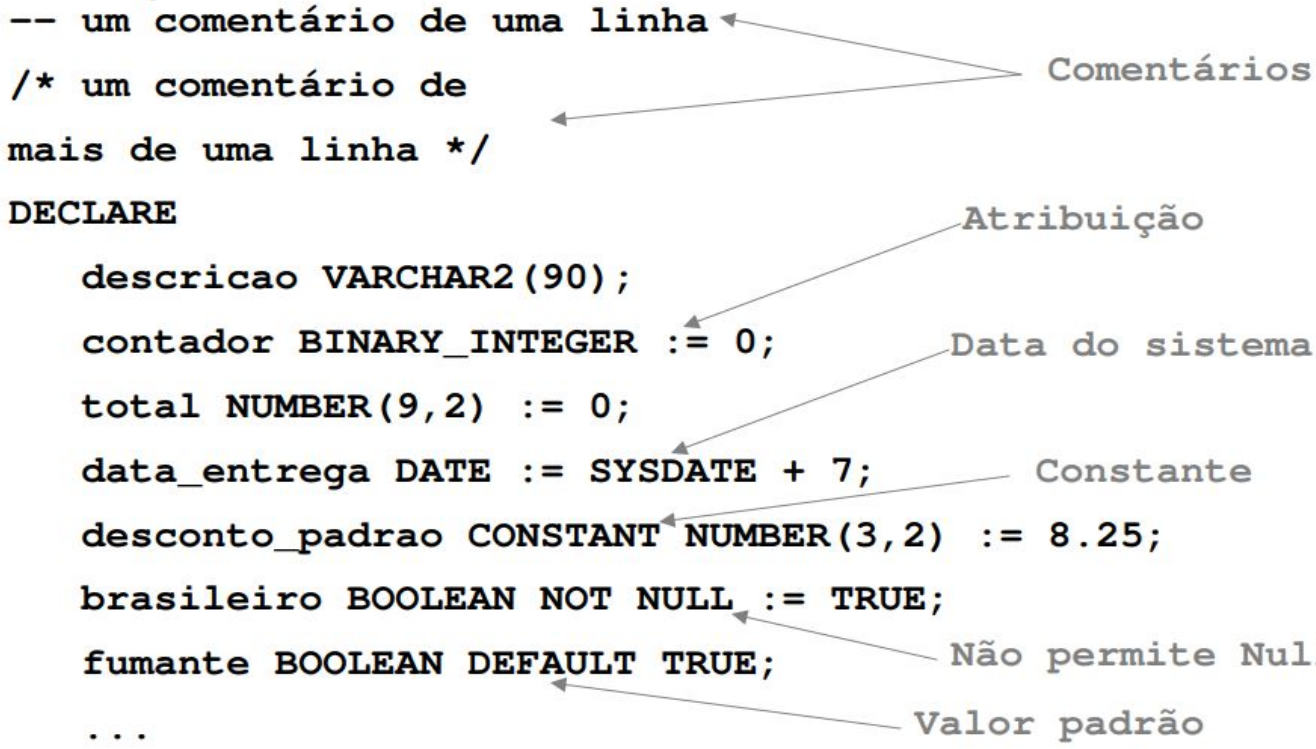
EXCEPTION (opcional)

/* define-se as ações que acontecem se uma exceção for lançado durante a execução deste bloco */

END; (obrigatório)

DECLARE

Exemplos



DECLARE

```
SET SERVEROUTPUT ON
DECLARE
    meuNome VARCHAR2(20);
BEGIN
    DBMS_OUTPUT.PUT_LINE('meu nome eh: '||meuNome);
    meuNome := 'Rita';
    DBMS_OUTPUT.PUT_LINE('meu nome eh: '||meuNome);
END;
/
```

ativa a saída de dados

imprime uma linha

Concatena string

Exemplo: Bloco Anônimo

```
-- Bloco Anonimo #1 --
-- Exibe o dia da semana da data corrente --
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hoje o dia da semana é: ' || TO_CHAR(SYSDATE,'DAY'));
END;

-- Bloco Anonimo #2 --
-- Exibe a data e hora correntes --
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Data/Hora: ' || TO_CHAR(SYSDATE, 'DD/MON/YYYY
                                    hh24:mm'));
END;
```

Declarando Variáveis com %TYPE

- Sintaxe:
  - identificador {tabela.coluna|variável}%TYPE;
- Exemplo:

```
DECLARE
    ...
    descricao produto.desc%TYPE;
    balanco NUMBER(7,2);
    resumo_balanco balanco%TYPE := 1000;
    ...
```

Declarando Variáveis com %TYPE

```
DECLARE
    nomeP pesquisador.nome%TYPE;
    instituicaoP pesquisador.instituicao%TYPE;
BEGIN
    SELECT nome,instituicao INTO nomeP,instituicaoP
    FROM pesquisador WHERE codPesquisador = 'p001';
    DBMS_OUTPUT.PUT_LINE('Nome,Inst: '||nomeP);
    DBMS_OUTPUT.PUT_LINE('Nome,Inst: '||instituicaoP);
END;
/
```

Criando um Record (Registro)

- Um **record** é um tipo de variável que podemos definir (como 'struct' em C ou 'object' em Java)

```
DECLARE
  TYPE sailor_record_type IS RECORD
    (sname          VARCHAR2(10),
     sid            VARCHAR2(9),
     age            NUMBER(3),
     rating         NUMBER(3));
  sailor_record       sailor_record_type;
...
BEGIN
  sailor_record.sname:='peter';
  sailor_record.age:=45;
...
```

Tipo %ROWTYPE

- Qualquer estrutura (e.g. cursores e nomes de tabela) que tem um tipo tupla pode ter seu tipo capturado com %ROWTYPE;
- Pode-se criar variáveis temporárias tipo tupla e acessar seus componentes como variável.componente (“dot notation”);
- Muito útil, principalmente se a tupla tem muitos componentes.

Declarando Variáveis com %ROWTYPE

- Declare uma variável (registro) com o tipo de uma linha de uma tabela.
- No exemplo abaixo: a variável reserves_record é do tipo da tupla da tabela Reserves

```
reserves_record   Reserves%ROWTYPE;
```

E como acessar os campos de reserves_record?

```
reserves_record.sid:=9;
Reserves_record.bid:=877;
```

Exemplo: Bloco Anônimo

```
-- Bloco Anonimo #3 --
-- Exibe dados de um funcionario da tabela empregado
DECLARE
    emp empregado%ROWTYPE;
BEGIN
     SELECT * INTO emp FROM empregado   WHERE mat= 1000;

     DBMS_OUTPUT.PUT_LINE('---------------------------------');
     DBMS_OUTPUT.PUT_LINE('ID: ' || TO_CHAR(emp.mat));
     DBMS_OUTPUT.PUT_LINE('Nome: ' || TO_CHAR(emp.nome));
     DBMS_OUTPUT.PUT_LINE('Salário: ' || TO_CHAR(emp.salario));
     DBMS_OUTPUT.PUT_LINE('---------------------------------');
END;
```

Condicionais

```
IF <condição> THEN
        <comando(s)>
ELSIF <condição> THEN
        <comandos(s)>
ELSE
        <comando(s)>
END IF;
```

```
CASE [ expressão]
   WHEN condição_1 THEN resultado_1
   WHEN condição_2 THEN resultado_2
   ...
   WHEN condição_n THEN resultado_n

   ELSE resultado

END
```

Exemplos

```
DECLARE
   x number:=10;
BEGIN
   IF x <> 0 THEN
      IF x < 0 THEN
         DBMS_OUTPUT.PUT_LINE(' X < 0 ');
      ELSE
         DBMS_OUTPUT.PUT_LINE(' X > 0 ');
      END IF;
   ELSE
      DBMS_OUTPUT.PUT_LINE(' X = 0 ');
   END IF;
END;
/
```

Exemplos

```
DECLARE
        p_ano int = 2010;
        studio char[15] = ‘Marvel’;
BEGIN
        IF NOT EXISTS (SELECT * FROM Filme WHERE ano = p_ano AND nomeStudio = studio) THEN
                DBMS_OUTPUT.PUT_LINE(‘NO’);
        ELSE;
                DBMS_OUTPUT.PUT_LINE(‘YES’);
        END IF;
END;
```

Cursores

- Ponteiro para uma única tupla do resultado da consulta (result set)
- Cada cursor possui uma consulta associada, especificada como parte da operação que define o cursor
- A consulta é executada quando o cursor for aberto
- Em uma mesma transação, um cursor pode ser aberto ou fechado qualquer número de vezes
- Pode-se ter vários cursores abertos ao mesmo tempo

Cursores

- Declaração
  CURSOR <nome> IS
  comando select-from-where

- O cursor aponta para cada tupla por vez da relação-resultado da consulta select-from-where, usando um *fetch statement* dentro de um laço.
  - Fetch statement:
- FETCH <nome_cursor> INTO
  lista_variáveis;
- Um laço é interrompido por:
  - EXIT WHEN <nome_cursor>%NOTFOUND;
  - O valor é TRUE se não houver mais tupla a apontar

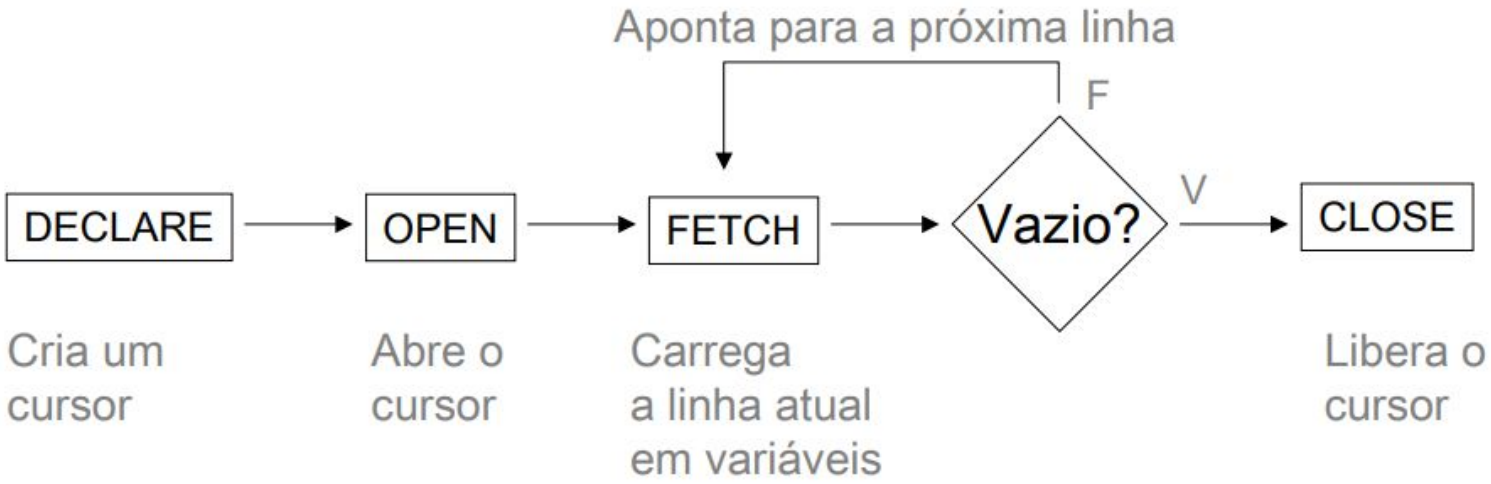
- OPEN e CLOSE abrem e fecham um cursor

Cursores

- Um cursor possui as seguintes operações:

- **OPEN**
  - Executa a consulta especificada e põe o cursor para apontar para uma posição anterior à primeira tupla do resultado da consulta
- **FETCH**
  - Move o cursor para apontar para próxima linha no resultado da consulta, tornando-a a tupla corrente e copiando todos os valores dos atributos para as variáveis da linguagem hospedeira usada
- **CLOSE**
  - Fecha o cursor

Cursores


Aponta para a próxima linha
F
DECLARE
OPEN
FETCH
Vazio?
V
CLOSE
Cria um cursor
Abre o cursor
Carrega a linha atual em variáveis
Libera o cursor

Cursores

- **Exemplo**:

```
DECLARE
        CURSOR c IS SELECT name, salary FROM customer;
        v_name customer.name%TYPE;
        v_salary customer.salary%TYPE;
BEGIN
        OPEN c;
        FETCH c INTO v_name, v_salary;
        ...
```

Cursores

```
DECLARE
    inst VARCHAR2(8):= 'UFPE';
    CURSOR pesq_cursor IS
    SELECT codPesquisador, nome FROM pesquisador
    WHERE instituicao = inst;
    codP pesquisador.codPesquisador%TYPE;
    nm pesquisador.nome%TYPE;
    ...
BEGIN
    OPEN pesq_cursor;
    LOOP
        FETCH pesq_cursor INTO codp, nm;
        EXIT WHEN pesq_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( codp ||' '||nm);
    END LOOP;
    CLOSE pesq_cursor;
    ...
END;
/
```

Cursores

- Por default, cursores movem-se do início do *result set* para frente (forward)
- Podemos movê-los também para trás e/ou para qualquer posição no *result set*
- Devemos acrescentar SCROLL na definição do cursor
- Exemplo
  - EXEC DECLARE meuCursor SCROLL CURSOR FOR Empregado;

Cursores

- Num FETCH, podemos adicionar as seguintes opções:
  - NEXT ou PRIOR: pega o próximo ou anterior
  - FIRST ou LAST: obtém o primeiro ou último
  - RELATIVE seguido de um inteiro: indica quantas tuplas mover para frente (se positivo) ou para trás (se negativo)
  - ABSOLUTE seguido de um inteiro: indica a posição da tupla contando do início (se positivo) ou do final (se negativo)

Exemplo

- **Procedimento que calcula a duração média dos filmes de um estúdio**

```
DECLARE
    CURSOR filmeCursor IS select duracao from Filme where nomeStudio = 'Disney';
    novaDuracao INTEGER;
    contaFilmes INTEGER;
    mean REAL;
BEGIN
        mean := 0.0;
        contaFilmes := 0;
        OPEN filmeCursor;
        LOOP
                FETCH filmeCursor INTO novaDuracao;
                EXIT WHEN filmeCursor%NOTFOUND;
                contaFilmes := contaFilmes + 1;
                mean := mean + novaDuracao;
        END LOOP;
        mean := mean / contaFilmes;
        CLOSE filmeCursor;
END;
```

Atributos explícitos de cursores

- Obtém informações de status sobre um cursor

Atributo	Tipo	Descrição
%ISOPEN	Boolean	Retorna TRUE se o cursor estiver aberto
%NOTFOUND	Boolean	Retorna TRUE se o fetch mais recente não retorna uma tupla
%FOUND	Boolean	Retorna TRUE se o fetch mais recente retorna uma tupla (complemento de %NOTFOUND)
%ROWCOUNT	Number	Retorna o total de tuplas acessadas até o momento

Iterações

Laços FOR

```
for <counter> in <lower_bound>..<higher_bound>
loop
   ...<<set of statements>>...
end loop;
```

Ex.:

```
BEGIN
    for i in 1..10
    loop
       DBMS_OUTPUT_put_line('i = ' || i);
    end loop;
END;
```

Exemplo

```
DECLARE
        contaFilmes INTEGER;
        mean REAL;


BEGIN
        mean := 0.0;
        contaFilmes := 0;
        FOR filme  IN (SELECT duracao FROM Filme WHERE nomeStudio = 'Disney');
        LOOP
                contaFilmes := contaFilmes + 1;
                mean := mean + duracao;
        END LOOP;
        mean := mean / contaFilmes;
END;
```

- **Obs.: veja que não precisa de OPEN, FETCH e CLOSE do cursor**

Cursor implícito com FOR

```
CREATE OR REPLACE PROCEDURE imprime_grandes_salarios
IS
BEGIN
        FOR emp_rec IN (SELECT last_name, salary FROM employees WHERE salary >= 50000) LOOP
            DBMS_OUTPUT.PUT_LINE ('O empregado '||emp_rec.last_name||' ganha '||
                                                emp_rec.salary||' reais por mês');
       END LOOP;
END;
```

Outros Tipos de Laços em PSM

- **Laço WHILE**

```
WHILE <condição> DO
        <comandos>
END WHILE;
```

- **REPEAT**

```
REPEAT
        <comandos>
UNTIL <condição>
END REPEAT;
```

Exemplo

```
DECLARE
    TEN number:=10;
    i number_table.num%TYPE:=1;
BEGIN
    WHILE i <= TEN LOOP
        INSERT INTO number_table VALUES (i);
        i := i + 1;
    END LOOP;
END;
```

Funções

- Podemos definir uma função da seguinte forma:

CREATE FUNCTION <name> (<param_list>)
RETURN <return_type>
IS ...

- No corpo da função, "RETURN <expression>;" sai (retorna) da função e retorna o valor de <expression>

Exemplos

```
CREATE FUNCTION P_Filmes (p_ano int, studio char[15])
RETURNS BOOLEAN
IS
BEGIN
      IF EXISTS (SELECT * FROM Filme WHERE ano = p_ano AND
            nomeStudio =  studio) THEN
                  RETURN TRUE;
      ELSE RETURN FALSE;
      END IF;
END;
```

Exemplos

```
CREATE FUNCTION get_descricao_produto (p_ID IN Produto.idprod%type)
RETURN Produto.descricao%type
IS
    v_descricao Produto.descricao%type;

BEGIN
      SELECT descricao INTO v_descricao FROM Produto WHERE idprod = p_ID;
      RETURN v_descricao
END;

/* invocando a função acima a partir de um select */

SELECT idprod, get_descricao_produto(idprod) FROM Produto WHERE idprod = 1;
```

Exemplos

```
DECLARE
   v_desc Produto.Descricao%type;
   v_id     Produto.IDPROD%type := 2;

BEGIN
    v_desc := get_descricao_produto (v_id);
    dbms_output_put_line (´A descrição do produto é' || v_desc);
END;
```

Tratamento de erros

- **Sintaxe**

```
EXCEPTION
  WHEN    nomeExceção1   THEN
        Comandos;
  WHEN    nomeExceção2   THEN
        Comandos;
  WHEN          others              THEN
        Comandos;
```

1. Oracle **tem códigos de erro próprios** (como `ORA-01403`, `ORA-06512`, etc.) que são reservados para o sistema.
2. Para **erros personalizados criados pelo usuário**, a Oracle **reserva a faixa de números negativos de -20000 até -20999**.
   - `-20000` → primeiro código disponível para erro customizado
   - `-20999` → último código permitido

Exemplos

- **Exemplo**

```
CREATE TABLE Pais (id NUMBER PRIMARY KEY, nome VARCHAR2(20));

BEGIN
        INSERT INTO pais VALUES (100, 'Brasil');
        COMMIT;
         DBMS_OUTPUT.PUT_LINE('Inserção realizada com sucesso');
        EXCEPTION
                WHEN dup_val_on_index THEN
                        DBMS_OUTPUT.PUT_LINE('País já cadastrado! ');
                WHEN others THEN
                        DBMS_OUTPUT.PUT_LINE('Erro ao cadastrar país');
END;
```

Tratando Exceções

O que é uma exceção?

- **Exceção** = situação anormal ou erro que ocorre durante a execução de um bloco PL/SQL.

- Exemplo: divisão por zero, registro não encontrado, violação de chave única, ou erros criados pelo próprio programador.

Tratando Exceções

- **Estrutura básica do tratamento de exceções**

  Todo bloco PL/SQL pode (deve) ter:

```
BEGIN
   -- Código principal
EXCEPTION
   WHEN <exceção> THEN
        -- Código para tratar o erro
END;
```

- Assim, se ocorrer um erro dentro do `BEGIN`, o controle passa para o `EXCEPTION`.

Tratando Exceções

Tipos de Exceção:

1. **Exceções pré-definidas (nativas do Oracle)**
   - Ex.: `NO_DATA_FOUND`, `ZERO_DIVIDE`, `DUP_VAL_ON_INDEX`.
2. **Exceções definidas pelo usuário**
   - Criadas pelo programador com `EXCEPTION`.
3. **Exceções associadas a códigos de erro específicos**
   - Associadas com `PRAGMA EXCEPTION_INIT`.

Tratando Exceções

Exemplo de Exceção Pré-definida:

```
DECLARE
   v_result NUMBER;
BEGIN
   v_result := 10 / 0;  -- Vai gerar divisão por zero
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Erro: divisão por zero não permitida!');
END;
```

Tratando Exceções

Exemplo de Exceção Definida pelo Usuário:

```
DECLARE
   e_idade_invalida EXCEPTION;  -- Criando exceção personalizada
   v_idade NUMBER := 15;
BEGIN
   IF v_idade < 18 THEN
      RAISE e_idade_invalida;   -- Dispara a exceção
   END IF;

EXCEPTION
   WHEN e_idade_invalida THEN
      DBMS_OUTPUT.PUT_LINE('Erro: idade deve ser maior ou igual a 18.');
END;
```

Tratando Exceções

RAISE_APPLICATION_ERROR ( ):

erro retornado ao Oracle com código

- Usado para **informar erros personalizados ao Oracle**, que podem ser tratados por sistemas externos.

- Faixa válida de códigos: **-20000 a -20999**.

Tratando Exceções

RAISE_APPLICATION_ERROR ( ):

```
BEGIN

   RAISE_APPLICATION_ERROR(-20001, 'Erro de validação:
valor inválido!');

END;
```

O ORACLE retorna:

ORA-20001: Erro de validação: valor inválido!

Tratando Exceções

Uso de uma exceção nomeada.

```
DECLARE
  e_idade EXCEPTION;
BEGIN
  RAISE e_idade;
EXCEPTION
  WHEN e_idade THEN
    DBMS_OUTPUT.PUT_LINE('CAPTURADO: Idade inválida!');
END;
```

Saída: Capturado: Idade inválida!

Tratando Exceções

Uso com

PRAGMA:

```
DECLARE
  e_idade EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_idade, -20010);
  v_idade NUMBER := -5;
BEGIN
  IF v_idade < 0 OR v_idade > 150 THEN
    RAISE_APPLICATION_ERROR(-20010, 'Idade deve estar entre 0 e 150 anos');
  END IF;

  -- Resto da lógica aqui...

EXCEPTION
  WHEN e_idade THEN
    DBMS_OUTPUT.PUT_LINE('ERRO DE VALIDAÇÃO: ' || SQLERRM);
END;
```

Exceções com raise_application_errors

```
DECLARE
    e_salario_neg EXCEPTION;
    e_idade_neg EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_salario_neg, -20020);
    PRAGMA EXCEPTION_INIT(e_idade_neg, -20021);
    v_salario NUMBER := -1000;
    v_idade NUMBER := -5;
BEGIN
    IF v_salario < 0 THEN
        RAISE_APPLICATION_ERROR(-20020, 'Salário não pode ser negativo');
    ELSIF v_idade < 0 THEN
        RAISE_APPLICATION_ERROR(-20021, 'Idade não pode ser negativa');
    END IF;

EXCEPTION
    WHEN e_salario_neg THEN
        DBMS_OUTPUT.PUT_LINE('Erro salário: ' || SQLERRM);
    WHEN e_idade_neg THEN
        DBMS_OUTPUT.PUT_LINE('Erro idade: ' || SQLERRM);
END;
```

Exemplos

```
DECLARE
   stock_price NUMBER := 9.73;
   net_earnings NUMBER := 0;
   pe_ratio NUMBER;
BEGIN
-- Calculation might cause division-by-zero error.
   pe_ratio := stock_price / net_earnings;
   dbms_output.put_line('Price/earnings ratio = ' || pe_ratio);

EXCEPTION  -- exception handlers begin

-- Only one of the WHEN blocks is executed.

   WHEN ZERO_DIVIDE THEN  -- handles 'division by zero' error
      dbms_output.put_line('Company must have had zero earnings.');
      pe_ratio := null;

   WHEN OTHERS THEN  -- handles all other errors
      dbms_output.put_line('Some other kind of error occurred.');
      pe_ratio := null;

END;  -- exception handlers and block end here
```

Exceções Pré-definidas

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Visualizando erros na criação de uma procedure/function

- Quando se cria uma procedure, se houver erros na sua definição, estes não serão mostrados
- Para ver os erros de procedure chamada myProcedure, digite:
- SHOW ERRORS PROCEDURE myProcedure
  - no iSQLPLUS prompt
- Para funções, digite:
- SHOW ERRORS FUNCTION myFunction

Stored Procedures

- São objetos armazenados no BD que usam comandos PL/SQL e SQL em seus corpos
- **Sintaxe**

```
CREATE OR REPLACE PROCEDURE  <nome> (<lista_argumentos>)
IS
    <declarações>
BEGIN
            <comandos PL/SQL e SQL>
END;
```

Stored Procedures

- <Lista_argumentos> tem triplas nome-modo-tipo.
  - Modo: IN, OUT ou IN OUT para read-only, write-only, read/write, respectivamente.
    - OBS.: Se omitido o Modo, por default é IN (parâmetro de entrada read-only)
  - Tipos de Dados
    - Padrão SQL + tipos genéricos como NUMBER = qualquer tipo inteiro ou real
  - Como tipos nas procedures devem casar com tipos no esquema do BD, pode-se usar uma expressão da forma
    - tabela.campo%TYPE
  - para capturar o tipo corretamente

Exemplos

- Uma procedure que inclui uma nova cerveja e seu preço no menu do bar AlviRubro
- VendeBeer (bar, cerveja, preço)

```
CREATE OR REPLACE PROCEDURE MenuAlviRubro (
                p_cerva IN VendeBeer.cerveja%TYPE,
                p_preco IN VendeBeer.preço%TYPE)
IS
BEGIN
         INSERT INTO VendeBeer VALUES('AlviRubro', p_cerva, p_preco);
        COMMIT;
END;
```

Exemplos

```
CREATE OR REPLACE PROCEDURE p1 (p_empid IN NUMBER, p_sal OUT NUMBER)
IS
BEGIN
        SELECT salary
        INTO       p_sal
        FROM     employees
        WHERE   employee_id = p_empid;
END;
```

```
CREATE OR REPLACE PROCEDURE p2
IS
         v_sal NUMBER;
         v_empid NUMBER := 101;
BEGIN
        p1(v_empid,v_sal);
        DBMS_OUTPUT.PUT_LINE('O empregado '||TO_CHAR(v_empid)||' recebe '||TO_CHAR(v_sal));
END;
```

Exemplos

- Procedure para incluir um Segmento de Mercado na tabela SEGMERCADO (ID, DESCRICAO)

```
CREATE OR REPLACE PROCEDURE incluir_segmercado (
                p_ID IN SEGMERCADO.ID%TYPE,
                p_DESCRICAO IN SEGMERCADO.DESCRICAO%TYPE)
IS
BEGIN
         INSERT INTO SEGMERCADO VALUES( p_ID,  UPPER(p_DESCRICAO));
        COMMIT;
END;
```

Removendo funções e procedures

- Para remover uma stored procedure/function:
  - DROP PROCEDURE <procedure_name>;
  - DROP FUNCTION <function_name>;

Como executar funções e procedures

FUNCTION

```
DECLARE
texto VARCHAR2(1000);
BEGIN
texto := func01(2);
DBMS_OUTPUT.PUT_LINE(texto);
END;
```

PROCEDURE

```
BEGIN
        MenuAlviRubro('AlviRubro','Bud', 2.50);
        MenuAlviRubro('AlviRubro','Carlsberg', 5.00);
END;
```

Fora de um bloco BEGIN ... END;

```
EXEC MenuAlviRubro('AlviRubro','Carlsberg', 5.00);
```

EXEMPLOS

Exibe Data e hora

```
CREATE OR REPLACE PROCEDURE mostrar_data
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Data atual: ' || TO_CHAR(SYSDATE, 'DD/MM/YYYY
HH24:MI:SS'));
END;
```

Exemplo com parâmetro IN

```
CREATE OR REPLACE PROCEDURE aumentar_salario (
    p_emp_id IN EMPREGADOS.ID%TYPE,
    p_percentual IN NUMBER
)
IS
BEGIN
    UPDATE EMPREGADOS
    SET salario = salario * (1 + p_percentual/100)
    WHERE id = p_emp_id;

    DBMS_OUTPUT.PUT_LINE('Salário atualizado com sucesso!');
END;
```

Exemplo com parâmetro OUT

```
CREATE OR REPLACE PROCEDURE obter_salario (
    p_emp_id IN EMPREGADOS.ID%TYPE,
    p_salario OUT EMPREGADOS.SALARIO%TYPE
)
IS
BEGIN
    SELECT salario
    INTO p_salario
    FROM EMPREGADOS
    WHERE id = p_emp_id;
END;
```

EXECUÇÃO:

```
VARIABLE v_sal NUMBER;
EXEC obter_salario(101, :v_sal);
PRINT v_sal;
```

Exemplo com os parâmetros IN e OUT

```
CREATE OR REPLACE PROCEDURE calcular_bonus (
    p_emp_id IN EMPREGADOS.ID%TYPE,
    p_bonus OUT NUMBER
)
IS
    v_salario EMPREGADOS.SALARIO%TYPE;
BEGIN
    SELECT salario INTO v_salario
    FROM EMPREGADOS
    WHERE id = p_emp_id;

    p_bonus := v_salario * 0.15; -- 15% de bônus
END;
```

EXECUÇÃO:

```
VARIABLE v_bonus NUMBER;
EXEC calcular_bonus(101, :v_bonus);
PRINT v_bonus;
```

Exemplo com CURSOR

```
CREATE OR REPLACE PROCEDURE
listar_empregados
IS
    CURSOR c_emp IS
        SELECT id, nome, salario FROM EMPREGADOS;
BEGIN
    FOR r IN c_emp LOOP
        DBMS_OUTPUT.PUT_LINE(r.id || ' - ' || r.nome || '
- ' || r.salario);
    END LOOP;
END;
```

EXECUÇÃO:

```
SET SERVEROUTPUT ON;
EXEC listar_empregados;
```

Considere a tabela Cliente

- SEGMERCADO (ID, DESCRICAO)
- CLIENTE (ID, RAZAO_SOCIAL, CNPJ, SEGMERCADO_id, DATA_INCLUSAO, FATURAMENTO_PREVISTO, CATEGORIA)
  /* procedure para incluir um cliente */

```
CREATE OR REPLACE PROCEDURE incluir_cliente
(       p_ID            CLIENTE.ID%type,
        p_RAZAO         CLIENTE.RAZAO_SOCIAL%type,
        p_CNPJ          CLIENTE.CNPJ%type,
        p_SEGMERCADO  CLIENTE.SEGMERCADO%type,
        p_FATURAMENTO CLIENTE.FATURAMENTO%type,
        p_CATEGORIA         CLIENTE.CATEGORIA%type   )

IS

BEGIN

        INSERT INTO CLIENTE VALUES
                (p_ID, p_RAZAO, p_CNPJ, p_SEGMERCADO, p_FATURAMENTO, p_CATEGORIA);

        COMMIT;

END;
```

Considere a tabela Cliente

- Inserindo um cliente

EXECUTE incluir_cliente (1, ‘SUPERMERCADO CAMPEAO’, ‘1234567890’, 1, 9000, ‘MEDIO GRANDE’);

Obs.: Veja que não se passa a data como parâmetro

Obter a Categoria de um cliente

```
CREATE OR REPLACE FUNCTION categoria_cliente
(p_FATURAMENTO IN CLIENTE.FATURAMENTO_PREVISTO%type)
RETURN CLIENTE.CATEGORIA%type
IS
   v_CATEGORIA CLIENTE.CATEGORIA%type;
BEGIN
   IF p_FATURAMENTO <= 10000 THEN
     v_CATEGORIA := 'PEQUENO';
  ELSIF p_FATURAMENTO <= 50000 THEN
     v_CATEGORIA := 'MÉDIO';
  ELSIF p_FATURAMENTO <= 100000 THEN
     v_CATEGORIA := 'MÉDIO GRANDE';
  ELSE
     v_CATEGORIA := 'GRANDE';
  END IF;
  RETURN v_CATEGORIA;
END;
```

Obter a Categoria de um cliente

```
DECLARE
    c_categoria  CLIENTE.categoria%type;
BEGIN
    c_categoria := categoria_cliente(90000);
    DBMS_OUTPUT.PUT_LINE('Categoria : ' || c_categoria);
END;
```

Modificando incluir_cliente

```
CREATE OR REPLACE PROCEDURE incluir_cliente
(
p_ID CLIENTE.ID%type,
p_RAZAO CLIENTE.RAZAO_SOCIAL%type,
p_CNPJ CLIENTE.CNPJ%type,
p_SEGMERCADO CLIENTE.SEGMERCADO_ID%type,
p_FATURAMENTO CLIENTE.FATURAMENTO_PREVISTO%type
)
IS
  v_CATEGORIA CLIENTE.CATEGORIA%type;
BEGIN

   v_CATEGORIA := categoria_cliente(p_FATURAMENTO);

   INSERT INTO CLIENTE
   VALUES
   (p_ID, p_RAZAO, p_CNPJ, p_SEGMERCADO, SYSDATE, p_FATURAMENTO, v_CATEGORIA);
   COMMIT;
END;
```

Cálculo de Imposto

```
create or replace FUNCTION RETORNA_IMPOSTO
    (p_COD_PRODUTO produto_venda_exercicio.cod_produto%type)
    RETURN produto_venda_exercicio.percentual_imposto%type
IS
     v_CATEGORIA produto_exercicio.categoria%type;;
     v_IMPOSTO produto_venda_exercicio.percentual_imposto%type;
BEGIN
     v_CATEGORIA := retorna_categoria(p_COD_PRODUTO);
     IF TRIM(v_CATEGORIA) = 'Sucos de Frutas' THEN
        v_IMPOSTO := 10;
    ELSIF TRIM(v_CATEGORIA) = 'Águas' THEN
              v_IMPOSTO := 20;
   ELSIF TRIM(v_CATEGORIA) = 'Mate' THEN
             v_IMPOSTO := 15;
   END IF;
   RETURN v_IMPOSTO;
END;
```

Formata CNPJ (hipotético 999/99-9999)

```
CREATE OR REPLACE PROCEDURE FORMATA_CNPJ
(p_CNPJ IN OUT cliente.cnpj%type)
IS
BEGIN
    p_CNPJ := SUBSTR(p_CNPJ, 1,3) || '/' || SUBSTR(p_CNPJ, 4,2) || '-' || SUBSTR(p_CNPJ,6);
END;
```

Formata CNPJ (hipotético 999/99-9999)

Testando:

```
SET SERVEROUTPUT ON;
DECLARE
  v_CNPJ cliente.cnpj%type;
BEGIN
  v_CNPJ := '1234567890';
  dbms_output.put_line(v_CNPJ);
  FORMATA_CNPJ (v_CNPJ);
  dbms_output.put_line(v_CNPJ);
END;
```

Modificando o incluir_cliente()

```
CREATE OR REPLACE PROCEDURE incluir_cliente (
p_ID CLIENTE.ID%type,
p_RAZAO CLIENTE.RAZAO_SOCIAL%type,
p_CNPJ CLIENTE.CNPJ%type,
p_SEGMERCADO CLIENTE.SEGMERCADO_ID%type,
p_FATURAMENTO CLIENTE.FATURAMENTO_PREVISTO%type )
IS
 v_CATEGORIA CLIENTE.CATEGORIA%type;
 v_CNPJ CLIENTE.CNPJ%type ;= p_CNPJ;
BEGIN
  v_CATEGORIA := categoria_cliente(p_FATURAMENTO);
  FORMATA_CNPJ(v_CNPJ);
  INSERT INTO CLIENTE   VALUES  (p_ID, p_RAZAO, v_CNPJ, p_SEGMERCADO, SYSDATE, p_FATURAMENTO,
                                 v_CATEGORIA);
  COMMIT;
END;
```

Procedure atualizar_segmercado()

```
CREATE OR REPLACE PROCEDURE ATUALIZAR_SEGMERCADO
(p_ID CLIENTE.ID%type, p_SEGMERCADO_ID CLIENTE.SEGMERCADO_ID%type)
IS
BEGIN
  UPDATE CLIENTE SET SEGMERCADO_ID = p_SEGMERCADO_ID WHERE ID = p_ID;
  COMMIT;
END;
```

Incovando atualizar_segmercado()

```
DECLARE
  v_SEGMERCADO CLIENTE.SEGMERCADO_ID%type := 4;
  v_NUMCLI INTEGER;
BEGIN
  SELECT COUNT(*) INTO v_NUMCLI FROM CLIENTE;
  FOR v_ID IN 1..v_NUMCLI LOOP
    ATUALIZAR_SEGMERCADO (v_ID,v_SEGMERCADO);
  END LOOP;
END;
```

Incovando atualizar_segmercado()

```
DECLARE
  v_SEGMERCADO CLIENTE.SEGMERCADO_ID%type := 4;
  v_NUMCLI INTEGER;
BEGIN
  SELECT COUNT(*) INTO v_NUMCLI FROM CLIENTE;
  FOR v_ID IN 1..v_NUMCLI LOOP
    ATUALIZAR_SEGMERCADO (v_ID,v_SEGMERCADO);
  END LOOP;
END;
```

Invocando atualizar_segmercado() – for

```
/*Mesmo anterior usndo WHILE ao invés de FOR */
DECLARE
  v_SEGMERCADO CLIENTE.SEGMERCADO_ID%type := 3;
  v_ID CLIENTE.ID%type := 1;
  v_NUMCLI INTEGER;
BEGIN
  SELECT COUNT(*) INTO v_NUMCLI FROM CLIENTE;
  WHILE v_ID <= v_NUMCLI LOOP
    ATUALIZAR_SEGMERCADO (v_ID,v_SEGMERCADO);
    v_ID := v_ID + 1;
  END LOOP;
END;
```

Invocando atualizar_segmercado() – while

```
/*Mesmo anterior usndo WHILE ao invés de FOR */
DECLARE
  v_SEGMERCADO CLIENTE.SEGMERCADO_ID%type := 3;
  v_ID CLIENTE.ID%type := 1;
  v_NUMCLI INTEGER;
BEGIN
  SELECT COUNT(*) INTO v_NUMCLI FROM CLIENTE;
  WHILE v_ID <= v_NUMCLI LOOP
    ATUALIZAR_SEGMERCADO (v_ID,v_SEGMERCADO);
    v_ID := v_ID + 1;
  END LOOP;
END;
```

Usando cursor com LOOP

```
DECLARE
  v_SEGMERCADO CLIENTE.SEGMERCADO_ID%type := 3;
  v_ID CLIENTE.ID%type;
  CURSOR cur_CLIENTE IS SELECT ID FROM CLIENTE;
BEGIN
  OPEN cur_CLIENTE;
  LOOP
    FETCH cur_CLIENTE INTO v_ID;
  EXIT WHEN cur_CLIENTE%NOTFOUND;
    ATUALIZAR_SEGMERCADO(v_ID, v_SEGMERCADO);
  END LOOP;
  CLOSE cur_CLIENTE;
END;
```

Usando cursor com WHILE

```
DECLARE
  v_SEGMERCADO CLIENTE.SEGMERCADO_ID%type := 3;
  v_ID CLIENTE.ID%type;
  CURSOR cur_CLIENTE IS SELECT ID FROM CLIENTE;
BEGIN
  OPEN cur_CLIENTE;
  FETCH cur_CLIENTE INTO v_ID;
  WHILE cur_CLIENTE%FOUND LOOP
    ATUALIZAR_SEGMERCADO(v_ID, v_SEGMERCADO);
    FETCH cur_CLIENTE INTO v_ID;
  END LOOP;
  CLOSE cur_CLIENTE;
END;
```

Usando cursor com FOR

```
DECLARE
  v_SEGMERCADO CLIENTE.SEGMERCADO_ID%type := 1;
BEGIN
  FOR linha_cur_CLIENTE IN (SELECT ID FROM CLIENTE) LOOP
    ATUALIZAR_SEGMERCADO (linha_cur_CLIENTE.ID, v_SEGMERCADO);
  END LOOP;
END;
```