PROGRAMAÇÃO EM
BANCO DE DADOS

APRESENTAÇÃO *da disciplina*

MILTON GOYA

1

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 APRESENTAÇÃO DA DISCIPLINA

1.1 PL/SQL

Olá! Seja bem-vindo. Iremos ver como usar estruturas que não são do escopo do SQL puro como definição de constantes e variáveis, estruturas de decisão, estruturas de repetição e tratamentos de exceção.

Veremos alguns conceitos de PL/SQL e como declarar seus identificadores. Veremos como é feita a interação do SQL com o PL/SQL, quais são os principais tipos de dados e como trabalhar com o controle transacional usando os comandos COMMIT, ROLLBACK e SAVEPOINT.

No capítulo que trata das estruturas de controle do PL/SQL, veremos como trabalhar com laços e estruturas condicionais.

Também veremos um recurso que permite que o SQL e o PL/SQL trabalhem com vários registros retornados: o CURSOR. Veremos tanto o cursor explícito quanto o cursor implícito.

Tratamento de exceção também é assunto. O tratamento de exceção pode ser usado para tratar erros detectados pelo Oracle ou definidos pelo programador.

Ainda veremos como tratar os objetos compilados pelo banco de dados Oracle.

Tem capítulo dedicado a procedimentos armazenados, ou STORAGE PROCEDURE. Um procedimento armazenado é um conjunto de instruções que pode ser compilado e armazenado no banco de dados. Isso permite que o conjunto de instruções possa ser compartilhado por vários programas, melhorando a eficiência do banco.

Veremos outro tipo de procedimento armazenado chamado função, ou FUNCTION. O que diferencia um tipo do outro é que a função sempre retorna um valor, enquanto o outro tipo pode, ou não, retornar um valor.

Teremos capítulo apresentando o conceito de pacote, ou PACKAGE. Um pacote nada mais é que um conjunto de procedimentos e/ou funções armazenados no banco de dados.

Por fim, teremos um capítulo dedicado aos gatilhos, ou TRIGGERS. Gatilhos são procedimentos que podem ser gravados em Java, PL/SQL ou C. São executados (ou disparados) implicitamente quando uma tabela é modificada, um objeto é criado ou ocorrem algumas ações de usuário ou de sistema de banco de dados.

Veremos as várias maneiras de trabalharmos com cursores, como identificarmos e manipularmos objetos compilados e como trabalhar com SQL dinâmico e coleções.

Também iremos ver como criar, alterar e excluir objetos armazenados como *storage procedures*, *functions*, *packages* e *triggers.* Iremos ver alguns elementos que afetam o desempenho de nossos programas e como poderemos analisar o seu custo de execução.

A todo momento estaremos vendo exemplos práticos que permitirão compreender melhor os conceitos teóricos.

1.2 Histórico

Feuerstein e Pribyl nos contam que, até 1991, só era possível criar estruturas procedurais com o SQL usando a linguagem PRO*C. O processo era trabalhoso, primeiro o programador escrevia um programa na linguagem C, depois, embutia os comandos SQL dentro desse programa. O código na linguagem C era pré-compilado para transformar as instruções SQL em bibliotecas e, só então, o programa era compilado e podia começar a ser testado. Esse processo tomava muito tempo dos desenvolveres e precisava ser melhorado.

O PL/SQL surgiu como uma resposta a uma solicitação da comunidade de desenvolvedores de uma linguagem de quarta geração, de alto nível, com foco em consulta a banco de dados. Foi lançado em 1991 junto com a versão 6.0 do Oracle e, na época, era uma extensão opcional que poderia ser incorporado ao banco de dados. Essa primeira versão era bem limitada e só na versão 2.0 passou a trabalhar com procedimentos armazenados (*storage procedures*), pacotes (*packages*) e funções (*functions*).

O significado do nome PL/SQL é "*Procedural Language extensions to SQL*". É muito similar ao SQL, mas incorpora funcionalidades de outras linguagens de programação, o que o torna uma poderosa linguagem para manipulação de dados.

Os desenvolvedores da linguagem PL/SQL tomaram como base a sintaxe da linguagem ADA, uma linguagem desenvolvida para o Departamento de Defesa dos Estados Unidos. A sintaxe da linguagem ADA, por sua vez, foi baseada na linguagem PASCAL.

Hoje em dia a linguagem PL/SQL é incorporada a diversas ferramentas Oracle e é linguagem básica para o desenvolvimento de programas complexos e poderosos.

1.3 Preparando o ambiente

Em nossos exemplos, iremos usar duas tabelas que são fornecidas pela própria Oracle para serem usadas em ambientes de teste e treinamento. As tabelas são EMP (Empregados) e DEPT (Departamentos). Nos meus exemplos, estou usando o usuário SCOTT. O usuário SCOTT é um usuário de treinamento fornecido pela própria Oracle. Você pode usar o seu usuário, sem problema algum. Outras tabelas serão criadas durante o curso, mas, por enquanto, essas serão o suficiente.

Uma curiosidade: Bruce Scott é um dos fundadores da Oracle, em 1977. É ele o criador do usuário SCOTT e das tabelas que usaremos em nossos exemplos durante o curso.

O primeiro passo para a construção do nosso ambiente de aprendizado é garantir que as tabelas DEPT e EMP estão consistentes. Antes de criar uma tabela nova, eu sempre tento apagá-la, isso garante que a nova tabela não irá entrar em conflito com outra que já exista em nosso ambiente. Para apagar uma tabela de maneira definitiva, usamos o comando DROP.

Execute os comandos abaixo no SQLDEVELOPER para garantir que as tabelas que iremos criar não existam em seu ambiente. Não estranhe se a execução do comando retornar a mensagem: **ORA-00942: tabela ou view não existe**. Essa mensagem só indica que a tabela – realmente – não existia no seu ambiente.

```
DROP TABLE dept PURGE;
DROP TABLE emp PURGE;
```

Código-fonte 1.1 – Eliminando possíveis tabelas pré-existentes no ambiente de estudo
Fonte: Oracle (1999)

Execute os comandos abaixo no SQLDEVELOPER para criar e popular a tabela DEPT. A tabela DEPT deve ser a primeira tabela a ser criada, pois existe uma chave estrangeira na coluna DEPTNO da tabela EMP que referencia a coluna DEPTNO da tabela DEPT.

```
CREATE TABLE dept(
  deptno NUMBER(2,0),
  dname  VARCHAR2(14),
  loc    VARCHAR2(13),
  CONSTRAINT pk_dept PRIMARY KEY (deptno)
);

INSERT INTO dept VALUES(10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO dept VALUES(20, 'RESEARCH', 'DALLAS');
INSERT INTO dept VALUES(30, 'SALES', 'CHICAGO');
INSERT INTO dept VALUES(40, 'OPERATIONS', 'BOSTON');
```

Código-fonte 1.2 – Eliminando possíveis tabelas pré-existentes no ambiente de estudo
Fonte: Oracle (1999)

A estrutura da tabela DEPT é a seguinte:

Coluna	Significado	Observação	Coluna
DEPTNO	Código do departamento	Numérico, inteiro com até 2 dígitos. Chave Primária	DEPTNO
DNAME	Nome do departamento	Alfanumérico com até 14 caracteres.	DNAME
LOC	Localização do departamento	Alfanumérico com até 13 caracteres.	LOC

Quadro 1.1 – Estrutura da tabela DEPT
Fonte: FIAP (2020)

Execute os comandos abaixo no SQLDEVELOPER para criar e popular a tabela EMP.

```
CREATE TABLE emp(
  empno    NUMBER(4,0),
  ename    VARCHAR2(10),
  job      VARCHAR2(9),
  mgr      NUMBER(4,0),
```

```
        hiredate DATE,
        sal      NUMBER(7,2),
        comm     NUMBER(7,2),
        deptno   NUMBER(2,0),
        CONSTRAINT pk_emp PRIMARY KEY (empno),
        CONSTRAINT fk_deptno FOREIGN KEY (deptno) REFERENCES
dept (deptno)
      );

      INSERT INTO EMP VALUES
              (7369, 'SMITH',  'CLERK',     7902,
              TO_DATE('17-DEZ-1980',  'DD-MON-YYYY'),    800,
NULL, 20);
      INSERT INTO EMP VALUES
              (7499, 'ALLEN',  'SALESMAN',  7698,
              TO_DATE('20-FEV-1981',   'DD-MON-YYYY'),   1600,
300, 30);
      INSERT INTO EMP VALUES
              (7521, 'WARD',   'SALESMAN',  7698,
              TO_DATE('22-FEV-1981',   'DD-MON-YYYY'),   1250,
500, 30);
      INSERT INTO EMP VALUES
              (7566, 'JONES',  'MANAGER',   7839,
              TO_DATE('2-ABR-1981',  'DD-MON-YYYY'),    2975,
NULL, 20);
      INSERT INTO EMP VALUES
              (7654, 'MARTIN', 'SALESMAN',  7698,
              TO_DATE('28-SET-1981',   'DD-MON-YYYY'),   1250,
1400, 30);
      INSERT INTO EMP VALUES
              (7698, 'BLAKE',  'MANAGER',   7839,
              TO_DATE('1-MAI-1981',  'DD-MON-YYYY'),    2850,
NULL, 30);
      INSERT INTO EMP VALUES
              (7782, 'CLARK',  'MANAGER',   7839,
              TO_DATE('9-JUN-1981',  'DD-MON-YYYY'),    2450,
NULL, 10);
      INSERT INTO EMP VALUES
              (7788, 'SCOTT',  'ANALYST',   7566,
              TO_DATE('09-DEZ-1982',   'DD-MON-YYYY'),   3000,
NULL, 20);
      INSERT INTO EMP VALUES
              (7839, 'KING',   'PRESIDENT', NULL,
              TO_DATE('17-NOV-1981',   'DD-MON-YYYY'),   5000,
NULL, 10);
      INSERT INTO EMP VALUES
              (7844, 'TURNER', 'SALESMAN',  7698,
              TO_DATE('8-SET-1981',  'DD-MON-YYYY'),    1500,
0, 30);
      INSERT INTO EMP VALUES
              (7876, 'ADAMS',  'CLERK',     7788,
```

```
                TO_DATE('12-JAN-1983',   'DD-MON-YYYY'),   1100,
NULL, 20);
      INSERT INTO EMP VALUES
                (7900, 'JAMES',  'CLERK',     7698,
                TO_DATE('3-DEZ-1981',  'DD-MON-YYYY'),      950,
NULL, 30);
      INSERT INTO EMP VALUES
                (7902, 'FORD',   'ANALYST',   7566,
                TO_DATE('3-DEZ-1981',  'DD-MON-YYYY'),     3000,
NULL, 20);
      INSERT INTO EMP VALUES
                (7934, 'MILLER', 'CLERK',     7782,
                TO_DATE('23-JAN-1982',   'DD-MON-YYYY'),   1300,
NULL, 10);
```

Código-fonte 1.3 – Criando e incluindo dados na tabela EMP
Fonte: Oracle (1999)

A estrutura da tabela EMP é a seguinte:

Coluna	Significado	Observação	Coluna
EMPNO	Código do empregado	Numérico, inteiro com até 4 dígitos. Chave Primária	EMPNO
ENAME	Nome do empregado	Alfanumérico com até 10 caracteres.	ENAME
JOB	Cargo do empregado	Alfanumérico com até 9 caracteres.	JOB
MGR	Código do gerente	Numérico, inteiro com até 4 dígitos.	MGR
HIREDATE	Data de contratação	Data	HIREDATE
SAL	Salário	Numérico, cinco dígitos inteiros e duas casas decimais	SAL
COMM	Comissão	Numérico, cinco dígitos inteiros e duas casas decimais	COMM
DEPTNO	Código do departamento	Numérico, inteiro com até 2 dígitos. Chave Estrangeira	DEPTNO

Quadro 1.2 - Estrutura da tabela EMP
Fonte: FIAP (2020)

Para verificar se a tabela DEPT e a tabela EMP foram criadas corretamente, execute o comando:

```
      SELECT table_name FROM user_tables;
```

Código-fonte 1.4 – Verificando se todas as tabelas foram criadas usando USER_TABLES
Fonte: Oracle (1999)

OU

```
      SELECT * FROM tab;
```

Código-fonte 1.5 – Verificando se todas as tabelas foram criadas usando TAB
Fonte: Oracle (1999)

O resultado esperado da execução dos dois comandos é similar à Figura Lista das tabelas existentes no usuário corrente:

```
SCOTT>select * from tab;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
BONUS                          TABLE
DEPT                           TABLE
EMP                            TABLE
SALGRADE                       TABLE

SCOTT>SELECT table_name FROM user_tables;

TABLE_NAME
------------------------------
DEPT
EMP
BONUS
SALGRADE
```

Figura 1.1 - Lista das tabelas existentes no usuário corrente
Fonte: Oracle (1999)

Note que você pode ter uma lista de tabelas diferentes das exibidas na Figura “Lista das tabelas existentes no usuário corrente”. O importante é que as tabelas DEPT e EMP tenham sido criadas.

Para verificar se o conteúdo das tabelas foi inserido corretamente, execute os seguintes comandos:

```
SELECT * FROM dept;
```

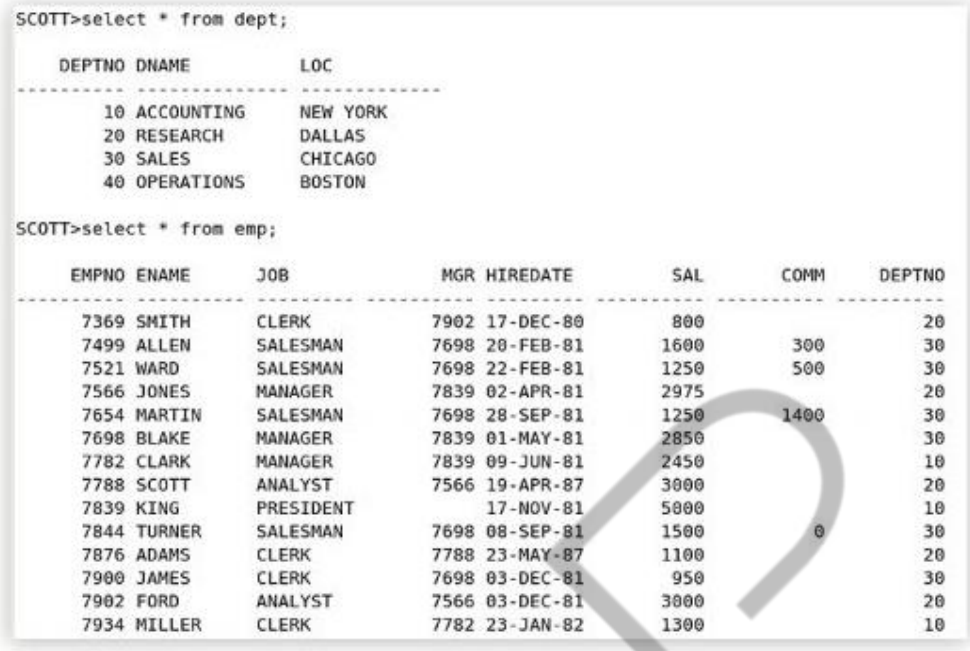
Código-fonte 1.6 – Verificando o conteúdo da tabela DEPT
Fonte: Oracle (1999)

E

```
SELECT * FROM emp;
```

Código-fonte 1.7 – Verificando o conteúdo da tabela EMP
Fonte: Oracle (1999)

O resultado esperado da execução dos dois comandos é similar à Figura “Resultado esperado da consulta as tabelas EMP e DEPT”, abaixo:



```
SCOTT>select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SCOTT>select * from emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10
```

Figura 1.2 - Resultado esperado da consulta às tabelas EMP e DEPT
Fonte: Oracle (1999)

Se não conseguir criar as tabelas ou, se tiver qualquer outra dificuldade no SQLDEVELOPER, poste sua dúvida no fórum.

1.4 Estrutura da linguagem PL/SQL

Para Dillon, et al. (2013), a linguagem PL/SQL é uma linguagem de programação de alto nível e possui vários recursos de programa como:

- Declaração de constantes e variáveis;
- Tipos de dados pré-definidos;
- Estruturas de seleção;
- Estruturas de repetição;
- Procedimentos;
- Funções;
- Pacotes.

Tudo isso pode ser desenvolvido em módulos ou, usando a terminologia da Oracle, em blocos. **Um bloco é a unidade básica de programação do PL/SQL.** Em

um bloco PL/SQL você pode usar comandos SQL, como o SELECT, INSERT, UPDATE e DELETE, e usar instruções como IF – THEN - ELSE.

Puga et al. (2015), nos conta que, com a linguagem PL/SQL, é possível criar blocos de anônimos ou blocos nomeados.

Um **bloco anônimo** é um conjunto de instruções que não fica armazenado definitivamente no banco. O conjunto de instruções é interpretado, executado e, mais tarde, descartado.

```
BEGIN
  NULL;
END;
```

Código-fonte 1.8 – Exemplo de bloco anônimo
Fonte: Fonte: PUGA et al. (2015)

Já um **bloco nomeado** é um conjunto de instruções que fica armazenado no banco de dados. Veremos esse assunto com mais detalhes ao longo dos capítulos.

Como já dissemos, a linguagem PL/SQL é estruturada em blocos e os programas podem ser divididos em blocos lógicos. Podemos dividir o bloco PL/SQL em quatro seções:

DECLARATIVA (opcional) - seção destinada à declaração das variáveis, cursores e exceções que serão utilizadas no bloco. É iniciado pela palavra DECLARE e, caso seja definida, deve ser a primeira estrutura do programa.

EXECUTÁVEL (obrigatória) - também conhecida como corpo do programa, área em que são descritos os passos necessários para realização da tarefa. Podem ser utilizadas instruções SQL e/ou PL/SQL. É iniciado pela palavra BEGIN.

TRATAMENTO DE EXCEÇÕES (opcional) - destinada ao tratamento das exceções geradas no bloco; devem ser descritas as ações a serem desempenhadas quando ocorrerem erros. É iniciado pela palavra EXCEPTION.

FIM (obrigatória) – destinado ao encerramento do programa. Todo programa PL/SQL deve ser finalizado usando a palavra END.

A estrutura de um bloco PL/SQL então, obedece a seguinte estrutura:

```
DECLARE (opcional)
```

```
      -- Aqui definimos as variáveis e outras estruturas que
veremos mais à frente
      BEGIN (obrigatório)
      -- Aqui usamos instruções SQL e PL/SQL
      EXCEPTION (opcional)
      -- Aqui definimos as ações que serão tomadas quando ocorrer
alguma exceção dentro do programa
      END; (obrigatório)
```

Quadro 1.3 – Estrutura de um Bloco PL/SQL
Fonte: PUGA et al. (2015)

Perceba que as palavras: DECLARE, BEGIN e EXCEPTION não são seguidas de ponto-e-vírgula, mas a palavra END; tem um ponto-e-vírgula no final. Lembre-se de colocar um ponto-e-vírgula ao final de todos os comandos SQL e PL/SQL usados em seu programa PL/SQL.

1.5 Blocos Anônimos

Para Puga et al. (2015) a linguagem PL/SQL permite criar blocos de anônimos ou blocos nomeados.

Reforçando o que dissemos anteriormente, os blocos anônimos não ficam armazenados na base de dados. Como consequência disso, segundo Puga et al. (2015) eles não podem ser chamados por outros blocos PL/SQL e eles devem ser compilados a cada utilização. É possível incorporar um bloco anônimo em uma aplicação ou executá-lo interativamente no SQL*Plus.

É importante observarmos que a recompilação de um programa a cada vez que é reexecutado consome recursos importantes de processamento e memória do servidor de banco de dados e pode impactar diretamente no desempenho geral do sistema.

Exemplo da estrutura de um bloco em PL/SQL:

```
      DECLARE
         v_variavel varchar2(5);
      BEGIN
         Select  nome_coluna
        into v_variavel
         from nome_tabela;
      EXCEPTION
         When exception_name then
      END;
```

```
/
```

Quadro 1.4 – Exemplo da Estrutura de um Bloco Anônimo
Fonte: PUGA et al. (2015)

Vamos analisar melhor o código do quadro 4:

- Na primeira linha aparece a palavra-chave DECLARE. Nesta seção iremos definir as constantes e variáveis que serão usadas no programa. Note que não há um ponto-e-vírgula após a palavra DECLARE.
- Na segunda linha estamos declarando o nome de uma variável chamada V_VARIAVEL. Após o nome da variável, informamos o tipo de dados que ela aceitará e, em seguida, o tamanho máximo dessa variável. Neste caso, V_VARIAVEL aceitará valores alfanuméricos pois foi definida com o tipo de dados VARCHAR2 com o tamanho de até cinco caracteres (5). Note que há um ponto-e-vírgula logo após a definição da variável.
- A terceira linha mostra a palavra-chave BEGIN. A seção executável inicia neste ponto. Também não há um ponto-e-vírgula após a palavra BEGIN.
- A quarta linha mostra o início de uma seleção, SELECT NOME_COLUNA, indicando que a coluna NOME_COLUNA será selecionada. Note que, como o comando não terminou, ainda não foi usado um ponto-e-vírgula.
- A quinta linha traz um comando específico do PL/SQL: INTO. O comando INTO V_VARIAVEL indica que o conteúdo da coluna NOME_DA_COLUNA será armazenada na variável V_VARIÁVEL. Ainda não terminamos de escrever o comando então ainda não colocaremos o ponto-e-vírgula.
- A sexta linha informa de qual tabela extrairemos os dados por meio do comando FROM NOME_TABELA. Desta vez colocaremos o ponto-e-vírgula para indicar que o comando encerrou.
- Na sétima linha aparece a palavra-chave EXCEPTION. Nesta seção iremos tratar as exceções encontradas durante a execução do programa. Note que não há um ponto-e-vírgula após a palavra Exception.
- A oitava linha mostra a sintaxe incompleta de como capturamos a exceção para tratá-la.

- A nona linha encerra o programa com a palavra-chave END;. Note que, desta vez temos um ponto-e-vírgula após a palavra END.
- A décima linha mostra uma / (barra). Quando estamos trabalhando com a interface SQL*Plus é esse caractere que encerra o editor PL/SQL e executa o bloco anônimo.
- Lembrando, as palavras-chave BEGIN e END são as únicas obrigatórias.

1.6 Variáveis

Variáveis são campos definidos pelo usuário e podem ser usados para armazenar dados temporariamente. (PUGA et al., 2015). Normalmente as variáveis são definidas e usadas em tempo de execução, isto é, são criadas quando o programa é executado e deixam de existir quando o programa é encerrado.

Ainda segundo PUGA et al. (2015), variáveis podem ser utilizadas para:

- **Manipulação de valores armazenados** - As variáveis podem ser usadas para cálculo e manipulação de outros dados sem acessar o banco de dados, ou seja, após os valores em memória não há necessidade de outros acessos para complemento da informação já armazenada.
- **Reutilização** - Quando declaradas, podem ser usadas repetidamente em uma aplicação simplesmente referenciando-as em outras instruções, incluindo outras instruções declarativas.
- **Facilitar a manutenção** – Pode-se declarar variáveis baseadas na estrutura das colunas das tabelas ou em outras variáveis (%TYPE e %ROWTYPE). Se uma definição subjacente for alterada, a declaração da variável é atualizada em tempo de execução. Isso permite a independência dos dados, reduz custos de manutenção e permite que os programas se adaptem de acordo com as alterações realizadas no banco de dados.
- **Passar valores** aos subprogramas PL/SQL por meio de parâmetros.
- **Exibir os resultados** em um bloco PL/SQL por meio de variáveis de saída.

A declaração e a inicialização das variáveis são feitas na seção declarativa de qualquer subprograma pacote ou bloco PL/SQL. As declarações alocam espaço de armazenamento para um valor, especificam seus tipos de dados e nomeiam a localização de armazenamento para que se possa referenciá-los. As declarações poderão também atribuir um valor inicial e impor a restrição NOT NULL.

Ao atribuir novos valores às variáveis na seção executável, o valor existente da variável é substituído pelo novo e é necessário declarar uma variável antes de referenciá-la em outras instruções, incluindo outras instruções declarativas.

1.7 Tipo de Variáveis

Todas as variáveis PL/SQL têm um tipo de dados, o qual especifica um formato de armazenamento, restrições e uma faixa válida de valores. A linguagem PL/SQL suporta quatro categorias de tipos de dados (Oracle, 2016):

Escalares - armazenam um único valor. Os principais tipos de dados são aqueles que correspondem aos tipos de coluna nas tabelas do Oracle Server, por exemplo VACHAR2, NUMBER, CHAR, entre outros; a linguagem PL/SQL também suporta variáveis booleanas.

Compostos – comporta o armazenamento de diferentes valores. Os tipos compostos em PL/SQL são registro, tabelas e matrizes.

Referenciais - armazenam valores, chamados de indicadores, que designam outros itens de programa. Um exemplo de tipos referenciais é o REF CURSOR.

LOB (large object) - armazenam blocos de dados não estruturados (como, por exemplo, texto, imagens gráficas, videoclipes e formatos de arquivo para armazenar sons) de até 4 gigabytes em tamanho. Os tipos de dados LOB fornecem acesso eficiente, aleatório e em intervalos aos dados, podendo ser atributos de um tipo de objeto.

As variáveis LOB podem ser classificadas como:

O tipo de dados **CLOB (character large object, objeto grande de caractere)** é usado para armazenar blocos grandes de dados com caracteres de um único byte no banco de dados.

O tipo de dados **BLOB (binary large object, objeto grande binário)** é usado para armazenar objeto binários grandes no banco de dados em linha (dentro de uma linha de tabela) ou fora de linha (fora da linha de tabela).

O tipo de dados **BFILE (binary file, arquivo binário)** é usado para armazenar objetos grandes binários em arquivos do sistema operacional fora do banco de dados.

O tipo de dados **NCLOB (objeto grande de caractere do idioma nacional)** é usado para armazenar blocos grandes de dados NCHAR de byte único ou de bytes múltiplos de largura fixa no banco de dados, dentro e fora de linha.

Ao declararmos uma variável, usamos a seguinte sintaxe:

```
identificador [CONSTANT] tipo de dados [NOT NULL]
          [:= valor para inicialização | expr default]
```

Quadro 1.5 – Sintaxe da declaração de variável
Fonte: ORACLE (2016)

Onde:

identificador é o nome da variável; Os identificadores não devem ter mais de 30 caracteres. O primeiro caractere deve ser uma letra; os demais podem ser letras, números ou símbolos especiais; não devem ser palavras reservadas nem haver espaços entre os caracteres.

CONSTANT restringe as variáveis para que o seu valor não possa ser alterado; as constantes devem ser inicializadas.

tipo de dados são tipos de dados escalares, compostos, referenciais ou LOB.

NOT NULL indica preenchimento obrigatório (variáveis NOT NULL devem ser inicializadas.)

expr é uma expressão PL/SQL que pode ser uma literal, uma outra variável ou uma expressão que envolve operadores e funções.

Para inicializar a variável, utiliza-se operador de atribuição (:=) ou a palavra reservada DEFAULT. Se não for atribuído um valor inicial, a nova variável conterá NULL por default.

Não é aconselhável identificar uma variável com nome igual ao nome das colunas de tabela usadas no bloco. Se as variáveis PL/SQL ocorrerem nas instruções SQL e tiverem o mesmo nome que uma coluna, o Oracle Server (2016) supõe que seja a coluna que esteja sendo referenciada.

1.8 Tipo dados escalares

Um tipo de dados escalares armazena um valor único e não possui componentes internos. Os tipos de dados escalares podem ser classificados em quatro categorias: número, caractere, data e booleano. Os tipos de dados de caractere e número possuem subtipos que associam um tipo básico a uma restrição. Por exemplo, INTEGER e POSITIVE são subtipos do tipo básico NUMBER. (Oracle, 2016). Segue abaixo exemplo de tipos de Dados Escalares Básicos:

Tipo de dado	Descrição
VARCHAR2 (tamanho)	Armazena caracteres com tamanho variável com até 32.767 bytes; não há tamanho default para estas variáveis
NUMBER (tamanho, precisão)	Armazena números reais ou inteiros
DATE	Armazena datas e horas entre os períodos de 4712 A.C. e 9999 D.C.
CHAR (tamanho)	Armazena caracteres de tamanho fixo até 32.767 bytes; tamanho default 1 byte.
BOOLEAN	Armazena um de três possíveis valores: TRUE, FALSE ou NULL
BINARY_INTEGER	Armazena números inteiros entre -2.147.483.647 e 2.147.483.647
PLS_INTEGER	Armazena números inteiros entre -2.147.483.647 e 2.147.483.647; estes valores requerem menos armazenamento e são mais rápidos que os valores NUMBER e BINARY_INTEGER

Quadro 1.6 – Principais dados escalares
Fonte: ORACLE (2016)

Note que existem mais tipos de dados que os exibidos no quadro. Por enquanto iremos trabalhar com esses. Sinta-se livre para pesquisar sobre os demais tipos de dados.

Vamos fazer alguns testes. Copie e cole o próximo código no SQLDeveloper e veja o resultado.

```
SET SERVEROUTPUT ON
DECLARE
    v_teste VARCHAR2(30):='Hello, World';
```

```
BEGIN
          DBMS_OUTPUT.PUT_LINE(v_teste);
END;
      /
```

Código-fonte 1.9 – Testando a definição de variáveis
Fonte: Fonte: PUGA et al. (2015), adaptado

Vamos analisar o código:

- Na linha 1 temos SET SERVEROUTPUT ON, esse comando informa que as mensagens do programa devem ser exibidas na tela. Vamos voltar a esse comando mais à frente.
- A linha 2 inicia a declaração de variáveis com a palavra-chave DECLARE.
- A linha 3 define uma variável denominada V_TESTE que aceita dados alfanuméricos com até 30 bytes de extensão. O operador := (dois ponto igual) atribui o valor ‘Hello, World’ a variável.
- A linha 4 inicia a seção executável com a palavra-chave BEGIN.
- A linha 5 exibe o conteúdo da variável V_TESTE na tela usando o pacote DBMS_OUTPUT.PUT_LINE. Voltaremos a esse pacote mais à frente.
- A linha 6 encerra o programa com a palavra-chave END.
- A linha 7 indica que o programa terminou e pode ser compilado e executado. O caractere barra (/) é usando em ambientes SQL*Plus.

Alguns exemplos de declaração de variáveis:

```
v_nascimento DATE;
v_data DATE := SYSDATE + 7;
v_codigo NUMBER(2) NOT NULL := 10;
v_UF VARCHAR2(2) := 'SP';
v_loc VARCHAR2(2) DEFAULT 'RJ';
v_teste_logico BOOLEAN := (v_valor1 < v_valor2);
c_const CONSTANT NUMBER := 54;
```

Código-fonte 1.10 – Exemplos de declaração de variáveis
Fonte: PUGA et al. (2015)

Explicando cada uma das declarações.

- **V_NASCIMENTO -** Declaração de variável do tipo data. A variável aceita valores nulos e não tem nenhum valor inicial atribuído a ele.

- **V_DATA -** Declaração de variável do tipo data. A inicialização é feita a partir de uma operação aritmética, conterá a data de hoje acrescida de sete dias.
- **V_CODIGO -** Declaração da variável do tipo numérico. Seu preenchimento é obrigatório por conta do NOT NULL. A variável está recebendo o valor inicial de 10.
- **V_UF -** Declaração e inicialização da variável com o valor SP; observe que os literais devem ser informados entre aspas simples (‘ ‘).
- **V_LOC -** Declaração da variável cujo valor default é RJ.
- **V_TESTE_LOGICO -** Declaração da variável booleana. A variável será inicializada com o resultado da expressão (v_valor1 < v_valor2).
- **C_CONST -** Declaração da constante numérica que está sendo inicializada.

1.9 Instruções SELECT em PL/SQL

Podemos usar a instrução SELECT para recuperar dados do banco de dados. A instrução SELECT precisa usar uma estrutura denominada CURSOR para recuperar mais de uma linha por vez, veremos essa estrutura um pouco mais à frente no curso. Por enquanto nossas consultas retornarão apenas uma linha (ORACLE, 2016). A sintaxe das instruções SELECT no PL/SQL é:

```
SELECT  colunas
INTO      {variáveis...| registro}
FROM      tabela
WHERE     condição;
```

Código-fonte 1.11 – Sintaxe da instrução SELECT no PL/SQL
Fonte: PUGA et al. (2015)

Onde:

- **Colunas -** lista de colunas que retornarão dados à consulta; podem incluir funções de linhas, de grupo ou expressões SQL
- **Into** - cláusula obrigatória, usada para especificar os nomes das variáveis que armazenarão os valores que o SQL retornará a partir da cláusula SELECT. Deve-se oferecer uma variável para cada coluna, seguindo a mesma ordem.

- **Variáveis** - nome das variáveis que irão armazenar o valor recuperado.
- **"Registro** - é o registro PL/SQL usado para armazenar os valores recuperados. Registros são tipos de dados compostos que podem ser utilizados para armazenar diversos tipos de informações como, por exemplo, colunas de uma tabela, expressões, constantes e variáveis PL/SQL.

```
SET SERVEROUTPUT ON
DECLARE
  v_nome    VARCHAR2(30);
  v_cargo   VARCHAR2(30);
BEGIN
  SELECT ename, job
  INTO v_nome, v_cargo
  FROM emp
  WHERE empno = 7934;
DBMS_OUTPUT.PUT_LINE(v_nome);
DBMS_OUTPUT.PUT_LINE(v_cargo);
END;
/
```

Código-fonte 1.12 – Recuperando o nome e o cargo do funcionário para o código indicado
Fonte: Fonte: PUGA et al. (2015), adaptado

Vamos analisar o código:

- Na linha 1 temos SET SERVEROUTPUT ON, esse comando informa que as mensagens do programa devem ser exibidas na tela. Vamos voltar a esse comando mais à frente.
- A linha 2 inicia a declaração de variáveis com a palavra-chave DECLARE.
- As linhas 3 e 4 definem as variáveis V_NOME e V_CARGO, alfanuméricas com até 30 bytes.
- A linha 5 inicia a seção executável com a palavra-chave BEGIN.
- As linhas 6, 7, 8 e 9 consultam o nome e cargo do funcionário 7934 e atribuem esses valores às variáveis V_NOME e V_CARGO, respectivamente.
- As linhas 10 e 11 exibem o conteúdo das variáveis V_NOME e V_CARGO na tela usando o pacote DBMS_OUTPUT.PUT_LINE.
- A linha 12 encerra o programa com a palavra-chave END;

- A linha 13 indica que o programa terminou e pode ser compilado e executado. O caractere barra (/) é usado em ambientes SQL*Plus.

```
      SET SERVEOUTPUT ON
      DECLARE
        v_soma_sal   NUMBER;
        v_deptno     NUMBER NOT NULL := 10;
      BEGIN
        SELECT SUM(sal)
        INTO        v_soma_sal
        FROM        emp
        WHERE  deptno = v_deptno;
      DBMS_OUTPUT.PUT_LINE('A soma dos salários do departamento
' || v_deptno || ' é ' || v_soma_sal);
      END;
      /
```

Código-fonte 1.13 – Calcular e exibir a somatória do salário de todos os funcionários de um determinado departamento
Fonte: Fonte: PUGA et al. (2015)

Vamos analisar o código:

- Na linha 1 temos SET SERVEROUTPUT ON, esse comando informa que as mensagens do programa devem ser exibidas na tela. Vamos voltar a esse comando mais à frente.
- A linha 2 inicia a declaração de variáveis com a palavra-chave DECLARE.
- As linhas 3 e 4 definem as variáveis V_SOMA_SAL e V_DEPTNO, numéricas. V_DEPTNO foi definido como NOT NULL então deve ter, obrigatoriamente, um valor inicial atribuído, nesse caso foi atribuído o valor de 10 para a variável.
- A linha 5 inicia a seção executável com a palavra-chave BEGIN.
- As linhas 6, 7, 8 e 9 somam o valor de todos os salários dos funcionários do departamento 10 e atribuem esse valor à variável V_SOMA_SAL.
- A linha 10 exibe o conteúdo das variáveis V_DEPTNO e V_SOMA_SAL na tela usando o pacote DBMS_OUTPUT.PUT_LINE.
- A linha 12 encerra o programa com a palavra-chave END;.

- A linha 13 indica que o programa terminou e pode ser compilado e executado. O caractere barra (/) é usando em ambientes SQL*Plus.

1.10 Instruções INSERT em PL/SQL

Podemos usar a instrução INSERT para incluir dados em uma tabela. A sintaxe dentro de um bloco PL/SQL é similar ao comando INSERT da linguagem SQL (ORACLE, 2016).

```
DECLARE
v_empno NUMBER := 11;
v_ename VARCHAR2(13) := 'SANDRA';
v_job VARCHAR2(13) := 'GERENTE';
v_deptno NUMBER := 10;
BEGIN
   INSERT INTO emp(empno, ename, job, deptno)
          VALUES (v_empno, v_ename, v_job, v_deptno);
END;
/
```

Código-fonte 1.14 – Adicionar informações sobre novos funcionários na tabela EMP
Fonte: Fonte: PUGA et al. (2015), adaptado por FIAP (2020)

Vamos analisar o código:

- A linha 1 inicia a declaração de variáveis com a palavra-chave DECLARE.
- As linhas 2, 3, 4 e 5 definem as variáveis V_EMPNO, V_ENAME, V_JOB e V_DEPTNO. Essas recebem os valores iniciais de 11, SANDRA, GERENTE e 10, respectivamente.
- A linha 6 inicia a seção executável com a palavra-chave BEGIN.
- As linhas 7 e 8 inserem os valores na tabela EMP usando o comando INSERT. Não há mudanças na sintaxe do comando SQL.
- A linha 9 encerra o programa com a palavra-chave END;.
- A linha 10 indica que o programa terminou e pode ser compilado e executado. O caractere barra (/) é usado em ambientes SQL*Plus.

1.11 Instruções UPDATE em PL/SQL

Podemos usar a instrução UPDATE para alterar dados em uma tabela. A sintaxe dentro de um bloco PL/SQL é similar ao comando UPDATE da linguagem SQL (ORACLE, 2016).

```
DECLARE
  v_sal_increase   NUMBER := 2000;
BEGIN
  UPDATE emp
  SET         sal = sal + v_sal_increase
  WHERE  job = 'ANALYST';
END;
/
```

Código-fonte 1.15 – Aumentar em 2000 o salário de todos os funcionários na tabela EMP que sejam analistas
Fonte: ORACLE (2016), adaptado

Vamos analisar o código:

- A linha 1 inicia a declaração de variáveis com a palavra-chave DECLARE.
- A linha 2 define a variável V_SAL_INCREASE, numérica. Essa recebe o valor inicial 2000.
- A linha 3 inicia a seção executável com a palavra-chave BEGIN.
- As linhas 4, 5, e 6 atualizam a tabela EMP, aumentando o salário de todos os funcionários que têm o cargo ANALYST em 2000, usando o comando UPDATE. Não há mudanças na sintaxe do comando SQL.
- A linha 7 encerra o programa com a palavra-chave END;.
- A linha 8 indica que o programa terminou e pode ser compilado e executado. O caractere barra (/) é usado em ambientes SQL*Plus.

1.12 Instruções DELETE em PL/SQL

Podemos usar a instrução DELETE para remover registros de uma tabela. A sintaxe dentro de um bloco PL/SQL é similar ao comando DELETE da linguagem SQL (ORACLE, 2016).

```
DECLARE
  v_deptno   NUMBER := 10;
BEGIN
  DELETE FROM   emp
  WHERE         deptno = v_deptno;
END;
/
```

Código-fonte 1.16 – Apagar todos os registros do departamento 10 da tabela EMP
Fonte: ORACLE (2016), adaptado

Vamos analisar o código:

- A linha 1 inicia a declaração de variáveis com a palavra-chave DECLARE.
- A linha 2 define a variável V_DEPTNO, numérica. Essa recebe o valor inicial de 10.
- A linha 3 inicia a seção executável com a palavra-chave BEGIN.
- As linhas 4 e 5 removem todos os registros do departamento 10 da tabela EMP usando o comando DELETE. Não há mudanças na sintaxe do comando SQL.
- A linha 6 encerra o programa com a palavra-chave END;.
- A linha 7 indica que o programa terminou e pode ser compilado e executado. O caractere barra (/) é usado em ambientes SQL*Plus.

1.13 Controle de Transações em PL/SQL

Uma transação consiste em uma ou mais instruções SQL, as quais, quando executadas, podem ser consideradas como uma única unidade; o que significa que, se uma instrução da transação falhar, a transação inteira falha e todas as instruções que foram executadas antes do ponto de falha são revertidas. (PUGA et al., 2015)

O início e término de uma transação definem os pontos da consistência da base de dados. Os efeitos de todas as operações SQL realizadas dentro de uma transação são aplicados ao banco de dados (COMMIT) ou os efeitos de todas as operações SQL realizadas são completamente "desfeitos" e jogados fora (ROLLBACK).

Em uma transação, *locks* ou bloqueios são obtidos no início, e mantidos ao longo da vida de uma transação. Quando surge uma condição de erro, o banco de

dados remove todas as alterações feitas pela transação. Em situações de erro, os bloqueios são liberados; nenhuma conexão é permitida até a consistência ser restaurada.

```
BEGIN
  INSERT INTO dept VALUES ('A','A','A');
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN ROLLBACK;
END;
/
```

Código-fonte 1.17 – Atualiza a tabela DEPT, em caso de falha desfaz as alterações
Fonte: ORACLE (2016), adaptado pelo autor (2020)

Vamos analisar o código:

- A linha 1 inicia a seção executável com a palavra-chave BEGIN.
- A linha 2 insere um novo registro na tabela DEPT. O primeiro campo da tabela DEPT tem o nome de DEPTNO e é numérico. Como estamos tentando inserir o valor ‘A’, que é alfanumérico, em um campo numérico, esperamos que o comando falhe.
- A linha 3 confirma as alterações com o comando COMMIT.
- A linha 4 inicia a seção de tratamento de exceções com a palavra-chave EXCEPTION.
- A linha 5 instrui o programa a executar o comando ROLLBACK, caso ocorra qualquer falha no programa. Veremos a instrução WHEN OTHERS mais adiante no curso.
- A linha 6 encerra o programa com a palavra-chave END;.
- A linha 7 indica que o programa terminou e pode ser compilado e executado. O caractere barra (/) é usando em ambientes SQL*Plus.

Como resultado, o programa é executado, mas nenhuma linha é inserida na tabela DEPT.

REFERÊNCIAS

DILLON, Sean; BECK, Christopher; KYTE, Thomas; KALLMAN, Joel; ROGERS, Howard. **Beginning Oracle Programming**. São Paulo: Apress, 2013.

FEUERSTEIN, Steven; PRIBYL, Bill. **Oracle Pl/Sql Programming**. California: O'Reilly Media, 2014.

ORACLE. **Oracle Concepts**. Oracle Press, 1999. Disponível em: <https://docs.oracle.com/cd/A87860_01/doc/server.817/a76965.pdf>. Acesso em: 26 ago. 2020.

_____. **Oracle Database: PL/SQL Language Reference 12c Release 2 (12.2)** B28370-05. Oracle Press, 2016.

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **Banco de dados**. São Paulo: Pearson, 2015.