Universidade Presbiteriana Mackenzie

Banco de Dados – Aula 19

Linguagem PL/SQL

Profa. Elisângela Botelho Gracias

Faculdade de Computação e Informática

Roteiro da Apresentação

- Introdução
- Estrutura de um Bloco PL/SQL
- Funcionamento do PL/SQL
- Integrando SQL em um programa PL/SQL

Roteiro da Apresentação

- Introdução
- Estrutura de um Bloco PL/SQL
- Funcionamento do PL/SQL
- Integrando SQL em um programa PL/SQL

Introdução

- **PL/SQL** é uma extensão da linguagem SQL
- Pode-se dizer que é uma **linguagem proprietária de programação de banco de dados do SGBD Oracle**

Introdução

- Por meio da **PL/SQL**, pode-se criar objetos de esquema, tais como:
  - *Stored procedures*
  - *Triggers*

Introdução

- *Stored procedures*
  - *Stored procedure*, ou procedimento armazenado, é um programa PL/SQL que pode ser acionado por uma aplicação, por um trigger ou uma ferramenta Oracle

Introdução

- *Triggers*
  - É um programa PL/SQL armazenado no banco de dados e que é executado imediatamente antes ou após os comandos INSERT, UPDATE e DELETE
  - A diferença principal entre um *trigger* e uma *procedure* está no fato de que as *procedures* são executadas através de uma chamada feita pelo usuário, enquanto os *triggers* são acionados pelo banco de dados

Roteiro da Apresentação

- Introdução
- Estrutura de um Bloco PL/SQL
- Funcionamento do PL/SQL
- Integrando SQL em um programa PL/SQL

Estrutura de um Bloco PL/SQL

- A linguagem PL/SQL utiliza o conceito de bloco estruturado
- A estrutura de um bloco PL/SQL é composta por uma área de declaração, uma área de comandos (execução) e uma área de exceções

Estrutura de um Bloco PL/SQL

```
[DECLARE
    declarações]
BEGIN
    estruturas executáveis (comandos) e outros blocos PL/SQL
    [BEGIN
    EXCEPTION
        tratamento de exceções (pode conter outros blocos)
   END;]
END;
```

Estrutura de um Bloco PL/SQL

- Seção de declaração (DECLARE): todos os objetos são declarados (é opcional)
- Seção de execução: os comandos PL/SQL são colocados
- Seção de exceção (EXCEPTION): os erros são tratados (é opcional)

Estrutura de um Bloco PL/SQL

- Seção de Declaração
- Seção de Execução
- Seção de Exceção

Estrutura de um Bloco PL/SQL

- Seção de Declaração
- Seção de Execução
- Seção de Exceção

Seção de Declaração

- A primeira seção do bloco PL/SQL, **seção de Declaração**, é **opcional**
- Contudo, se o bloco usar **variáveis ou constantes**, todas elas devem ser **previamente declaradas** antes de serem utilizadas em comandos

Seção de Declaração

- Esta seção é iniciada pela palavra-chave **DECLARE** e o desenvolvedor pode realizar as seguintes **tarefas**:
  - Declarar o nome de um identificador
  - Declarar o tipo do identificador (constante ou variável)
  - Declarar o tipo de dado do identificador
  - Atribuir (inicializar) um conteúdo ao identificador

Seção de Declaração
Variáveis e Constantes

- Alguns **observações importantes sobre variáveis e constantes**:
  - Devem começar com uma letra e ter no máximo 30 caracteres
  - Não podem ser idênticos aos nomes de tabelas ou nomes de atributos
  - Não podem ser idênticos a nenhuma palavra ou símbolo reservado

Seção de Declaração
Variáveis e Constantes

- As variáveis podem ter qualquer tipo de dado válido pela linguagem SQL e Oracle, como por exemplo:
  - **Alfanuméricos**: CHAR(N), VARCHAR(N), onde N é o comprimento máximo
  - **Numéricos**: INTEGER, FLOAT, REAL, NUMBER(W,D), DECIMAL(W,D), onde W é a largura e D é o número de dígitos à direita da vírgula decimal
  - **Data**: DATE, que armazena informações de data e hora

Seção de Declaração
Variáveis e Constantes

- Exemplo de declaração de uma variável:

```
DECLARE
        qtdade_itens INTEGER;
```

- É possível inicializar a variável, da seguinte forma:

```
DECLARE
        qtdade_itens INTEGER := 0;
```

- O comando := é o **comando de atribuição**

Seção de Declaração
Variáveis e Constantes

- A declaração de uma constante é parecida com a de uma variável, tendo apenas que adicionar a palavra-chave **CONSTANT** após o seu nome

```
DECLARE
    qtdade_itens INTEGER;
    valor_fixo CONSTANT INTEGER := 40000;
```

Seção de Declaração
Variáveis e Constantes

- Uma outra forma de atribuir o tipo de dado a uma variável é **herdando o tipo de dados de um atributo** de uma tabela da seguinte maneira:
  - nome_variável **nome_tabela.nome_atributo%type**

```
DECLARE
    qtdade_itens INTEGER;
    valor_fixo CONSTANT INTEGER := 40000;
    nome Funcionario.Nome_Func%type;
```

Estrutura de um Bloco PL/SQL

- Seção de Declaração
- Seção de Execução
- Seção de Exceção

Seção de Execução

- A **seção de execução** do bloco PL/SQL é iniciada com a declaração BEGIN
- Esta seção pode conter:
  - Comando **SQL**
  - Comandos de **controles lógicos**
  - Comandos de **atribuição**, dentre outros

Estrutura de um Bloco PL/SQL

- Seção de Declaração
- Seção de Execução
- Seção de Exceção

Seção de Exceção

- Na **seção de exceção** do bloco PL/SQL, o desenvolvedor pode usar comandos para tratar um erro que eventualmente ocorra durante a execução de um programa PL/SQL
- Pode-se criar uma rotina que execute procedimentos corretivos ao detectar um erro, evitando, assim, que o sistema fique interrompido

Roteiro da Apresentação

- Introdução
- Estrutura de um Bloco PL/SQL
- Funcionamento do PL/SQL
- Integrando SQL em um programa PL/SQL

Funcionamento do PL/SQL

- O PL/SQL executa os comandos procedurais e repassa os comandos SQL para o servidor Oracle processar
- A criação de blocos PL/SQL pode ser feita por meio de qualquer editor de texto
- Para executar um programa ou um script PL/SQL pode-se utilizar o SQL*Plus ou SQL Developer, que permite criar e executar blocos PL/SQL

Funcionamento do PL/SQL

- Como a maioria das linguagens procedurais, o PL/SQL possui comandos para controlar o fluxo de execução do programa
- São eles que fazem a diferença, realizando desvios, analisando condições e permitindo a tomada de decisões

Funcionamento do PL/SQL

- As principais **estruturas de controle** do PL/SQL podem ser divididas em:
  - Estruturas de **controles condicionais**
  - Estruturas de **controles sequenciais**
  - Estruturas de **controles de repetição ou iteração**

Funcionamento do PL/SQL
Estruturas de Controle

- Comando **IF** ... **THEN**
- Comando **LOOP**
- Comando **FOR** ... **LOOP**
- Comando **WHILE**
- Comando **CASE**

Funcionamento do PL/SQL
Estruturas de Controle

- Comando **IF** ... **THEN**
- Comando LOOP
- Comando FOR ... LOOP
- Comando WHILE
- Comando CASE

Estruturas de Controle
Comando IF ... THEN

- O comando **IF** ... **THEN** tem por função avaliar uma condição e executar uma ou mais linhas de comandos, somente se essa condição analisada for verdadeira
- Esse comando possui 2 variações

Estruturas de Controle
Comando IF ... THEN

- Sintaxe1

```
IF <condição> THEN
 <comando1>;
 <comandoN>;
END IF;
```

Estruturas de Controle
Comando IF ... THEN

- Os comandos que se encontram entre a cláusula THEN e END IF serão executados apenas se a <condição> for verdadeira

Estruturas de Controle
Comando IF ... THEN

- Exemplo:

```
IF (media_sal <= 1000) THEN
    UPDATE Empregado
    SET salario = salario * 2;
END IF;
```

Estruturas de Controle
Comando IF ... THEN

- Explicação do exemplo anterior:
  - media_sal é uma variável (que poderia ter recebido a média salarial dos funcionários)
  - se o valor desta variável media_sal for menor ou igual a 1000, então todos os funcionários receberão 100% de aumento

Estruturas de Controle
Comando IF ... THEN

- Sintaxe2

```
IF <condição> THEN
        <comando1>;
[ELSIF <condição2> THEN
        <comando2>;]
[ELSIF <condição3> THEN
        <comando3>;]
[ELSE
        <comando4>;]
END IF;
```

Estruturas de Controle
Comando IF ... THEN

- Na estrutura da sintaxe2, mais de uma condição pode ser analisada e, consequentemente, diversas ações podem ser executadas
- O PL/SQL usa a cláusula **ELSIF**, e não ELSEIF, como outras linguagens
- Um comando pode ter **inúmeras cláusulas ELSIF**, mas pode possuir apenas **uma cláusula ELSE**

Estruturas de Controle
Comando IF ... THEN

- Nesta estrutura, caso a condição principal for falsa, a primeira cláusula ELSIF será analisada e:
  - se for verdadeira, os comandos a seguir serão executados até que seja encontrada outra cláusula ELSIF ou ELSE
  - se for falsa, o programa testa a segunda, e assim sucessivamente. Ao encontrar uma condição verdadeira, são executados seus comandos, e o programa continua após a linha do comando END IF

Estruturas de Controle
Comando IF ... THEN

- Exemplo:

```
IF (media_sal < 2000) THEN
    UPDATE Empregado SET salario = salario * 1.4;
ELSIF (media_sal < 3000) THEN
    UPDATE Empregado SET salario = salario * 1.3;
ELSE
    UPDATE Empregado SET salario = salario * 1.2;
END IF;
```

Estruturas de Controle
Comando IF ... THEN

- Explicação do exemplo anterior:
  - media_sal é uma variável (que poderia ter recebido a média salarial dos funcionários)
  - se o valor desta variável media_sal for :
    - menor que 2000, então todos os funcionários receberão 40% de aumento
    - senão, se for menor que 3000, então todos os funcionários receberão 30% de aumento
    - senão, se nenhuma das condições anteriores forem verdadeiras, os funcionários receberão 20% de aumento

Estruturas de Controle
Comando IF ... THEN

- Pode-se utilizar os operadores lógicos - AND, OR e NOT:

```
IF (media_sal >= 800) AND (media_sal <=  1000) THEN
    UPDATE Empregado
    SET salario = salario * 1.3;
END IF;
```

Funcionamento do PL/SQL
Estruturas de Controle

- Comando IF ... THEN
- Comando LOOP
- Comando FOR ... LOOP
- Comando WHILE
- Comando CASE

Estruturas de Controle
Comando LOOP

- O comando **LOOP** inicializa um grupo de comandos indefinidamente ou até que uma condição force a "quebra" do loop e desvie a execução do programa para outro lugar
- Ele é usado em conjunto com o comando **EXIT**, responsável pela parada de execução do loop

Estruturas de Controle
Comando LOOP

- Sintaxe

```
LOOP
        <comandos>
        EXIT
        <comandos contendo EXIT>
 END LOOP;
```

Estruturas de Controle
Comando LOOP

- Sintaxe (outra versão)

```
LOOP
        <comandos>
        EXIT WHEN <condição>
        <comandos>
END LOOP;
```

Estruturas de Controle
Comando LOOP

- Exemplo:

```
i := 1;
LOOP
        i := i + 1;
        <comandos>
        IF (i >= 30) THEN
            EXIT;
        END IF;
        <comandos>
END LOOP;
```

Estruturas de Controle
Comando LOOP

- Exemplo (outra versão):

```
i := 1;
LOOP
        i := i + 1;
        <comandos>
        EXIT WHEN (i >= 30);
        <comandos>
END LOOP;
```

Funcionamento do PL/SQL
Estruturas de Controle

- Comando IF ... THEN
- Comando LOOP
- Comando FOR ... LOOP
- Comando WHILE
- Comando CASE

Estruturas de Controle
Comando FOR ... LOOP

- O comando **FOR ... LOOP** é uma variação do comando **LOOP**
- Aqui os comandos são executados automaticamente até que uma condição avaliada retorne falsa

Estruturas de Controle
Comando FOR ... LOOP

- O comando **FOR ... LOOP** executa um trecho de instruções de forma iterativa por meio de um contador com valor inicial e final
- Essa contagem pode ser crescente (**IN**) ou decrescente (**REVERSE**)

Estruturas de Controle
Comando FOR ... LOOP

- Sintaxe

```
FOR <contador> IN [REVERSE] <valor_inicial> ..
<valor_final>  LOOP
        <comandos>
END LOOP;
```

Estruturas de Controle
Comando FOR ... LOOP

- Exemplo:

```
FOR j IN 1 .. 10   LOOP
   <comandos>
END LOOP;
```

Estruturas de Controle
Comando FOR ... LOOP

- Neste exemplo, o comando FOR inicializa uma variável de controle - j - cujo valor inicial é 1
- Os <comandos> serão executados até encontrar o END LOOP e, nesse momento, o controle volta para o comando FOR, que incrementa a variável e analisa a condição mestra, ou seja, se o valor de j é menor que o valor final

Estruturas de Controle
Comando FOR ... LOOP

- A cláusula **REVERSE** faz com que o contador comece no valor mais alto e seja decrescido até atingir o valor mais baixo

Funcionamento do PL/SQL
Estruturas de Controle

- Comando IF ... THEN
- Comando LOOP
- Comando FOR ... LOOP
- Comando WHILE
- Comando CASE

Estruturas de Controle
Comando WHILE

- Outro controle de execução possível é o uso do comando **WHILE**
- Essa estrutura analisa uma condição e, somente se ela for verdadeira, executa os comandos contidos dentro dessa estrutura

Estruturas de Controle
Comando WHILE

- Sintaxe

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

Estruturas de Controle
Comando WHILE

- Exemplo:

```
x := 1;
WHILE (x < 20) LOOP
        <comandos>
        x := x+1;
END LOOP;
```

- Cuidado para não deixar o comando WHILE em um loop eterno
- Neste exemplo, a variável x foi incrementada para que isso não acontecesse

Funcionamento do PL/SQL
Estruturas de Controle

- Comando IF ... THEN
- Comando LOOP
- Comando FOR ... LOOP
- Comando WHILE
- Comando CASE

Estruturas de Controle
Comando CASE

- O comando **CASE** permite avaliar uma **expressão** e retornar um resultado entre várias alternativas disponíveis
- Ou seja, a instrução CASE utiliza um **seletor** em vez de uma condição
- **Seletor é uma expressão cujo valor determina a decisão**

Estruturas de Controle
Comando CASE

- Sintaxe

```
CASE <expressão>
   WHEN <expressão1> THEN <resultado1>
   WHEN <expressão2> THEN <resultado2>
   ...
   WHEN <expressãoN> THEN <resultadoN>
   [ELSE <resultadoN+1>]
END CASE;
```

Estruturas de Controle
Comando CASE

- **Exemplo**:

```
resultado :=
CASE nota
   WHEN 'A' THEN 'Excelente'
   WHEN 'B' THEN 'Muito Bom'
   WHEN 'C' THEN 'Bom'
   WHEN 'D' THEN 'Razoável'
   WHEN 'E' THEN 'Fraco'
   ELSE 'Sem chance'
END CASE;
```

Roteiro da Apresentação

- Introdução
- Estrutura de um Bloco PL/SQL
- Funcionamento do PL/SQL
- Integrando SQL em um programa PL/SQL

Integrando SQL em um programa PL/SQL

- Comandos da Linguagem SQL podem ser inseridos dentro da seção de execução de um bloco PL/SQL e executados quase sem diferença
- A **maior novidade** é que o **desenvolvedor pode usar**:
  - variável/constante declarada na seção de declaração, sendo que o conteúdo das variáveis pode ser manipulado pelos comandos SQL
  - estruturas de controles condicionais
  - estruturas de controles sequenciais
  - estruturas de controles de repetição ou iteração

Integrando SQL em um programa PL/SQL

- Criação da tabela Temp1 no seu esquema para utilizar as estruturas de controle e a criação de variáveis (utilize o SQL Developer):

```
CREATE TABLE Temp1
 (Codigo INTEGER,
  Data DATE,
  PRIMARY KEY(Codigo)
 );
```

Integrando SQL em um programa PL/SQL

- Este próximo exemplo insere linhas na tabela Temp, enquanto a variável i for menor ou igual a 10
- Em cada linha é inserida, para o atributo código da tabela Temp, o valor de i, e, para o atributo data, a data atual (sysdate)
- Observe, nos próximos exemplos, que é **fundamental a utilização do ; ao final de cada comando**

```
DECLARE
i INTEGER := 0;
BEGIN
   WHILE (i <= 10) LOOP
      INSERT
      INTO Temp1 (Codigo, Data)
      VALUES (i, sysdate);
   i := i + 1;
    END LOOP;
END;
```

```
DECLARE
i INTEGER := 0;
BEGIN
   WHILE (i <= 10) LOOP
      INSERT
      INTO Temp1 (Codigo, Data)
      VALUES (i, sysdate);
    i := i + 1;
    END LOOP;
END;
```

Integrando SQL em um programa PL/SQL

- Para ver o resultado utilize o comando SELECT:

```
SELECT *
FROM Temp1;
```

Integrando SQL em um programa PL/SQL

- Este próximo exemplo é bem semelhante ao anterior, só que é utilizado o comando **FOR** para executar o loop
- Sua **vantagem** é que não é **necessário controlar o incremento da variável de contador por meio da programação,** como deve ser feito no **WHILE**

```
DECLARE
i INTEGER:= 20;
BEGIN
   FOR i IN 20 .. 30 LOOP
     INSERT
     INTO Temp1 (Codigo, Data)
     VALUES (i, sysdate);
   END LOOP;
END;
```

```
DECLARE
i INTEGER:= 20;
BEGIN
  FOR i IN 20 .. 30 LOOP
     INSERT
     INTO Temp1 (Codigo, Data)
     VALUES (i, sysdate);
  END LOOP;
END;
```

Integrando SQL em um programa PL/SQL

- Para visualizar o resultado dos novos dados inseridos utilize o comando SELECT:

```
SELECT *
FROM Temp1
WHERE (codigo > 10);
```

Integrando SQL em um programa PL/SQL

- Este exemplo utiliza o exemplo anterior, mas acrescenta um teste condicional
- Ele insere apenas os registros cujo código seja múltiplo de 3
- Para isso, utilize a função **MOD**

```
DECLARE
i INTEGER := 40;
BEGIN
   FOR i IN 40 .. 50 LOOP
     IF (MOD(i,3)=0) THEN
       INSERT
       INTO Temp1 (Codigo, Data)
       VALUES (i, sysdate);
     END IF;
   END LOOP;
END;
```

```
DECLARE
i INTEGER := 40;
BEGIN
   FOR i IN 40 .. 50 LOOP
      IF (MOD(i,3)=0) THEN
         INSERT
         INTO Temp1 (Codigo, Data)
         VALUES (i, sysdate);
      END IF;
   END LOOP;
END;
```

Integrando SQL em um programa PL/SQL

- Para visualizar o resultado dos novos dados inseridos utilize o comando SELECT:

```
SELECT *
 FROM Temp1
WHERE (codigo > 30);
```

Integrando SQL em um programa PL/SQL

- Este exemplo também utiliza o exemplo anterior, mas o comando **IF** tem uma cláusula **ELSE**
- Se o valor de i for múltiplo de 3, o registro será inserido normalmente, com o valor de i e a data do sistema
- Caso contrário, é feito um cálculo para que seja gravado o valor de i*5 no atributo código e a data do sistema somada ao valor de i no atributo data, ou seja, a data atual somada com i dias

```
DECLARE
i INTEGER := 60;
BEGIN
   FOR i IN 60 .. 70 LOOP
      IF (MOD(i,3) = 0) THEN
         INSERT
         INTO Temp1 (Codigo, Data)
         VALUES (i, sysdate);
      ELSE
         INSERT
         INTO Temp1 (Codigo, Data)
         VALUES (i*5, sysdate + i);
      END IF;
   END LOOP;
END;
```

```
DECLARE
i INTEGER := 60;
BEGIN
   FOR i IN 60 .. 70 LOOP
      IF (MOD(i,3) = 0) THEN
         INSERT
         INTO Temp1 (Codigo, Data)
         VALUES (i, sysdate);
      ELSE
         INSERT
         INTO Temp1 (Codigo, Data)
         VALUES (i*5, sysdate + i);
      END IF;
   END LOOP;
END;
```

Integrando SQL em um programa PL/SQL

- Para visualizar o resultado dos novos dados inseridos utilize o comando SELECT:

```
SELECT *
FROM Temp1
WHERE (codigo >= 60);
```

Integrando SQL em um programa PL/SQL

- Este último exemplo insere um novo registro, sendo que para o valor do código a ser inserido, será buscado o maior valor para um código já existente na tabela Temp1 e somar um a este valor (para a data será utilizada a data atual do sistema)
- Para isso, será utilizado o **SELECT INTO...**

```
DECLARE
cod_max INTEGER := 0;
BEGIN

   SELECT MAX(Codigo) INTO cod_max
   FROM Temp1;

   INSERT
   INTO Temp1 (Codigo, Data)
   VALUES (cod_max + 1, sysdate);
END;
```

```
DECLARE
cod_max INTEGER := 0;
BEGIN

   SELECT MAX(Codigo) INTO cod_max
   FROM Temp1;

   INSERT
   INTO Temp1 (Codigo, Data)
   VALUES (cod_max + 1, sysdate);
END;
```

Uma outra forma de fazer o exemplo anterior, sem criar variáveis:

```
BEGIN
   INSERT
   INTO Temp1 (Codigo, Data)
   VALUES ((SELECT MAX(Codigo)+1
              FROM Temp1),
             sysdate);
END;
```

Uma outra forma de fazer o exemplo anterior, sem criar variáveis:

```
BEGIN
   INSERT
   INTO Temp1 (Codigo, Data)
   VALUES ((SELECT MAX(Codigo)+1
            FROM Temp1),
            sysdate);
END;
```

Integrando SQL em um programa PL/SQL

- Observe, no exemplo anterior, que **não foi utilizada a seção DECLARE,** pois não tinha nenhuma variável a ser definida
- O comando SELECT pode ser utilizado, diretamente, dentro do INSERT (observando que o resultado tem que ser somente um valor inteiro, neste caso)

```
SET SERVEROUTPUT ON;
DECLARE
   soma INTEGER := 0;
   produto INTEGER := 1;
   i INTEGER := 0;
BEGIN
   FOR i IN 1 .. 10 LOOP
      soma := soma + i;
      produto := produto * i;
   END LOOP;
   Dbms_Output.Put_Line('Soma: '|| soma);
   Dbms_Output.Put_Line('Produto: '||produto);
END;
```

```
SET SERVEROUTPUT ON;
DECLARE
   soma INTEGER := 0;
   produto INTEGER := 1;
   i INTEGER := 0;
BEGIN
   FOR i IN 1 .. 10 LOOP
      soma := soma + i;
      produto := produto * i;
   END LOOP;
   Dbms_Output.Put_Line('Soma: '|| soma);
   Dbms_Output.Put_Line('Produto: '||produto);
END;
```

Integrando SQL em um programa PL/SQL

- O comando **”SET SERVEROUTPUT ON;”** serve para habilitar a utilização do pacote Dbms_Output.Put_Line, que é utilizado para apresentação de mensagens.
- O procedimento **Dbms_Output.Put_Line** mostra o resultado, que no exemplo anterior, mostra:
  - a soma dos valores entre 1 e 10 (inclusive)
  - o produto dos valores entre 1 e 10 (inclusive)

Exercício

EXERCÍCIO

De acordo com o banco de dados, a seguir, desenvolva um bloco simples de PL/SQL anônimo que busque o maior salário pago na empresa, bem como o menor salário, a média salarial e a quantidade de funcionários da empresa e insira estes dados em uma tabela "Relatorio" (cujo script de criação se encontra logo abaixo).

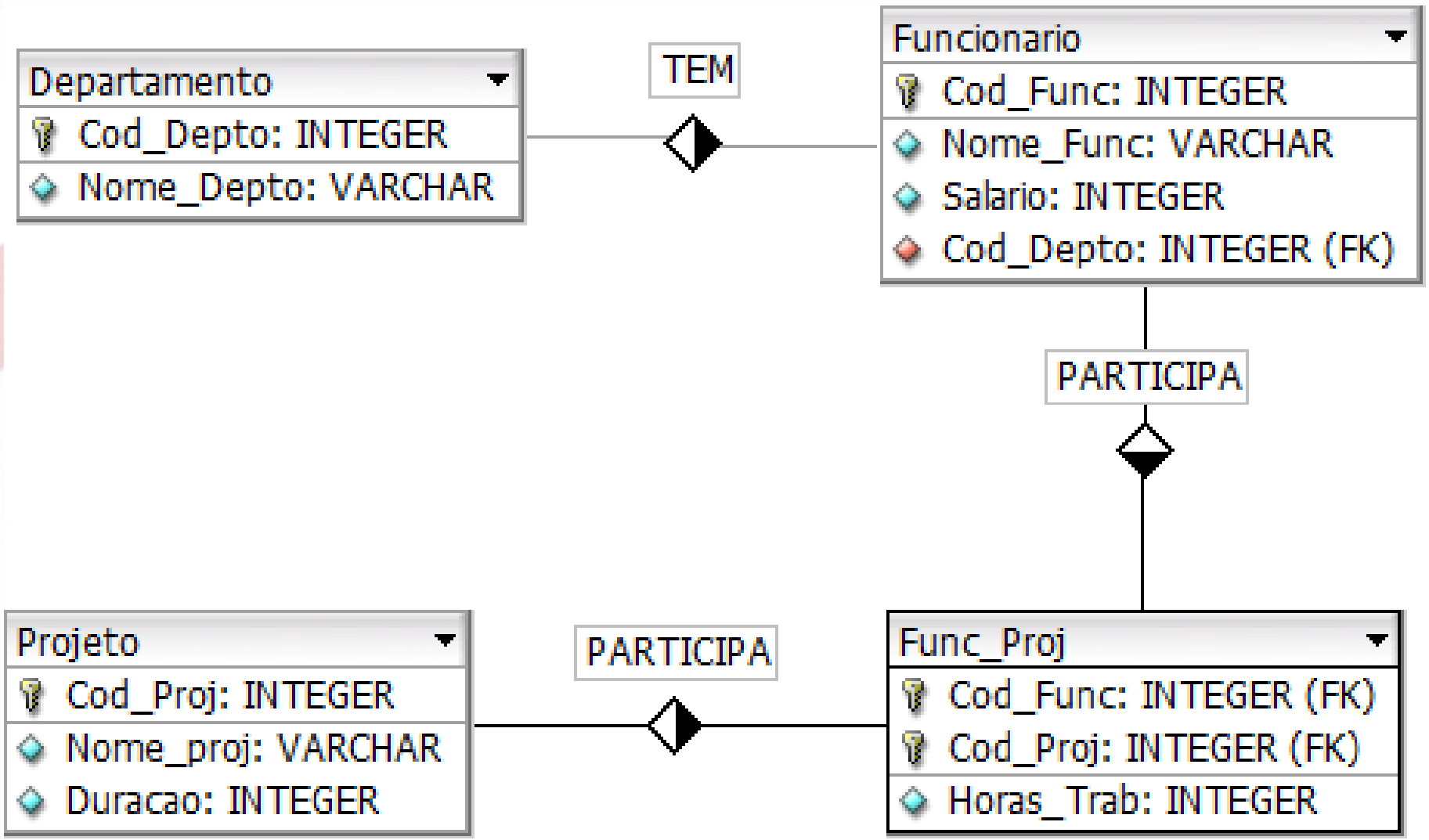
```
CREATE TABLE Relatorio
(Salario_Max INTEGER,
 Salario_Min INTEGER,
 Salario_Medio INTEGER,
 Nro_Funcionarios INTEGER
);
```

```
DECLARE
    maior_salario int := 0;
    menor_salario int := 0;
    media_salario int := 0;
    qtdade_func int := 0;
BEGIN
        SELECT MAX(Salario), MIN(Salario), AVG(Salario),
                COUNT(Cod_Func)
                INTO maior_salario, menor_salario,
                        media_salario, qtdade_func
        FROM Funcionario;

        INSERT INTO Relatorio
        VALUES (maior_salario, menor_salario, media_salario,
                 qtdade_func);
END;
```

EXERCÍCIO

- Para verificar se foi inserida uma linha na tabela Relatorio, execute a seguinte uma consulta:

```
SELECT *

FROM Relatorio;
```


Departamento
Cod_Depto: INTEGER
Nome_Depto: VARCHAR
TEM
Funcionario
Cod_Func: INTEGER
Nome_Func: VARCHAR
Salario: INTEGER
Cod_Depto: INTEGER (FK)
PARTICIPA
Projeto
Cod_Proj: INTEGER
Nome_proj: VARCHAR
Duracao: INTEGER
PARTICIPA
Func_Proj
Cod_Func: INTEGER (FK)
Cod_Proj: INTEGER (FK)
Horas_Trab: INTEGER

Departamento

Cod_Depto	Nome_Depto
1	Marketing
2	Vendas
3	Dados
4	Pesquisa

Funcionário

Cod_Func	Nome_Func	Salario	Cod_Depto
101	Joao da Silva	2000	2
102	Mario Souza	1500	1
103	Sergio Santos	2400	2
104	Maria Castro	1200	1
105	Marcio Santana	1400	4

Projeto

Cod_Proj	Nome_Proj	Duracao
1001	Sistema A	2
1002	Sistema B	6
1003	Sistema X	4

Func_Proj

Cod_Func	Cod_Proj	Horas_Trab
101	1001	24
101	1002	160
102	1001	56
102	1003	45
103	1001	86
103	1003	64
104	1001	46

```
-- Script de Criação do BD Projeto

DROP TABLE Func_Proj CASCADE CONSTRAINT;
DROP TABLE Projeto CASCADE CONSTRAINT;
DROP TABLE Funcionario CASCADE CONSTRAINT;
DROP TABLE Departamento CASCADE CONSTRAINT;

CREATE TABLE Departamento
(Cod_Depto INTEGER,
 Nome_Depto VARCHAR(20) NOT NULL,
 PRIMARY KEY(Cod_Depto));

CREATE TABLE Funcionario
(Cod_Func INTEGER,
 Nome_Func VARCHAR(20) NOT NULL,
 Salario INTEGER NOT NULL,
 Cod_Depto INTEGER NOT NULL,
 PRIMARY KEY(Cod_Func),
 FOREIGN KEY (Cod_Depto) REFERENCES Departamento (Cod_Depto));

CREATE TABLE Projeto
(Cod_Proj INTEGER,
 Nome_Proj VARCHAR(20) NOT NULL,
 Duracao INTEGER NOT NULL,
 PRIMARY KEY(Cod_Proj));

CREATE TABLE Func_Proj
(Cod_Func INTEGER,
 Cod_Proj INTEGER,
 Horas_Trab INTEGER,
 PRIMARY KEY(Cod_Func, Cod_Proj),
 FOREIGN KEY (Cod_Func) REFERENCES Funcionario(Cod_Func),
 FOREIGN KEY (Cod_Proj) REFERENCES Projeto(Cod_Proj));
```

```
INSERT INTO Departamento (Cod_Depto, Nome_Depto) VALUES (1, 'Marketing');
INSERT INTO Departamento (Cod_Depto, Nome_Depto) VALUES (2, 'Vendas');
INSERT INTO Departamento (Cod_Depto, Nome_Depto) VALUES (3, 'Dados');
INSERT INTO Departamento (Cod_Depto, Nome_Depto) VALUES (4, 'Pesquisa');

INSERT INTO Funcionario (Cod_Func, Nome_Func, Salario, Cod_Depto) VALUES (101, 'Joao da Silva Santos', 2000, 2);
INSERT INTO Funcionario (Cod_Func, Nome_Func, Salario, Cod_Depto) VALUES (102, 'Mario Souza', 1500, 1);
INSERT INTO Funcionario (Cod_Func, Nome_Func, Salario, Cod_Depto) VALUES (103, 'Sergio Silva Santos', 2400, 2);
INSERT INTO Funcionario (Cod_Func, Nome_Func, Salario, Cod_Depto) VALUES (104, 'Maria Castro', 1200, 1);
INSERT INTO Funcionario (Cod_Func, Nome_Func, Salario, Cod_Depto) VALUES (105, 'Marcio Silva Santana', 1400, 4);

INSERT INTO Projeto (Cod_Proj, Nome_Proj, Duracao) VALUES (1001, 'SistemaA', 2);
INSERT INTO Projeto (Cod_Proj, Nome_Proj, Duracao) VALUES (1002, 'SistemaB', 6);
INSERT INTO Projeto (Cod_Proj, Nome_Proj, Duracao) VALUES (1003, 'SistemaX', 4);

INSERT INTO Func_Proj (Cod_Func, Cod_Proj, Horas_Trab) VALUES (101, 1001, 24);
INSERT INTO Func_Proj (Cod_Func, Cod_Proj, Horas_Trab) VALUES (101, 1002, 160);
INSERT INTO Func_Proj (Cod_Func, Cod_Proj, Horas_Trab) VALUES (102, 1001, 56);
INSERT INTO Func_Proj (Cod_Func, Cod_Proj, Horas_Trab) VALUES (102, 1003, 45);
INSERT INTO Func_Proj (Cod_Func, Cod_Proj, Horas_Trab) VALUES (103, 1001, 86);
INSERT INTO Func_Proj (Cod_Func, Cod_Proj, Horas_Trab) VALUES (103, 1003, 64);
INSERT INTO Func_Proj (Cod_Func, Cod_Proj, Horas_Trab) VALUES (104, 1001, 46);
COMMIT;
```

Bibliografia

- MANNINO, M. V. *Projeto, Desenvolvimento de Aplicações & Administração de Banco de Dados*. São Paulo: McGraw-Hill, 2008.

Obrigado

Profa. Elisângela Botelho Gracias
elisangela.botelho@mackenzie.br