



“Programação para Banco de Dados” (Parte 1)

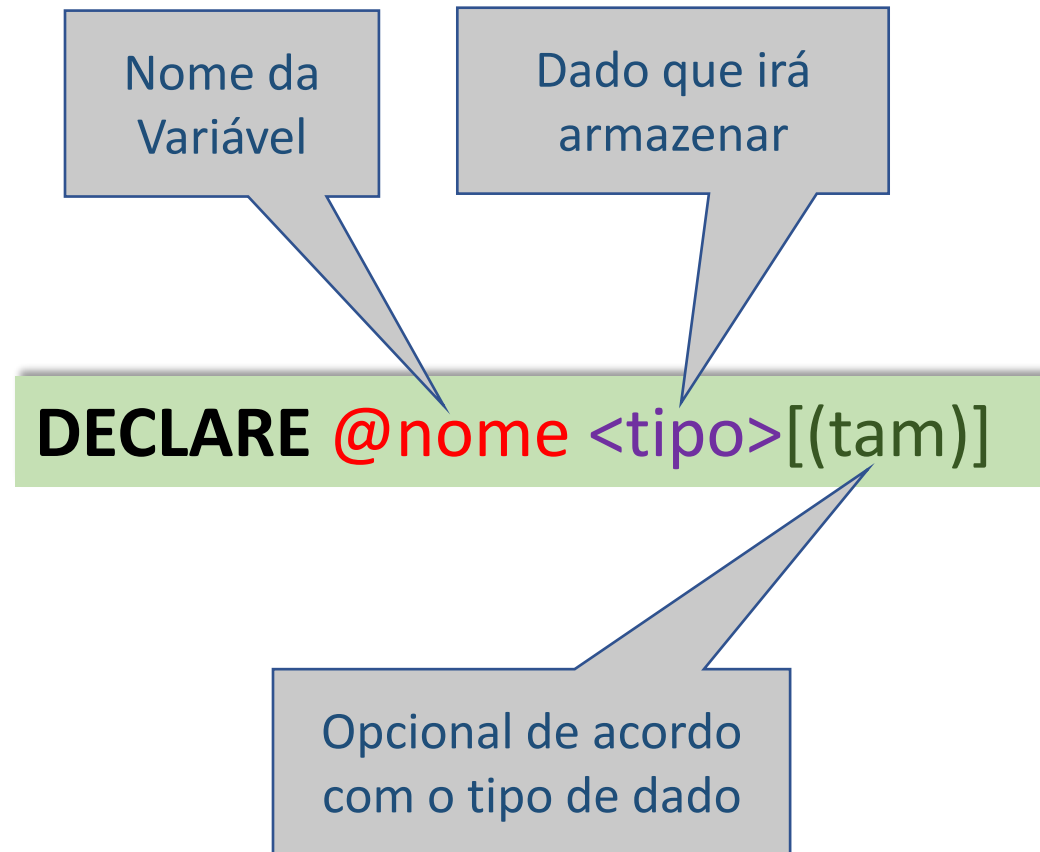
Introdução à Programação



- ☐ A **linguagem SQL** permite **obter e manipular dados**, mas ela também possui **elementos lógicos**;
- ☐ Os **DBA's** lidam com **scripts** que envolvem **mais do que simplesmente consultar dados**;
- ☐ Imagine ver a **data do último backup dos BD's** ou **gerenciar o espaço usado por cada um deles**;
- ☐ Assim como em outras linguagens é necessário trabalhar com **variáveis, repetição, seleção, etc.**

Declarando Variáveis

- ❑ Armazenam **valores temporários** que serão usados no programa;
- ❑ Para utilizá-las é necessário **declará-las**, ou seja, **especificá-las**;
- ❑ Toda variável sempre terá **um nome e o seu respectivo tipo**;
- ❑ Após a sua declaração, **valores poderão ser atribuídos à ela**.



Atribuindo Valor às Variáveis

- ❑ O modo mais comum é **inicializa-las** com um dado valor;
- ❑ O valor de uma variável é setado via declaração **SET**;
- ❑ O **SELECT** inicia múltiplas variáveis ao mesmo tempo;
- ❑ Outra situação é obter seus valores a partir de um **registro**.

Inicia a variável com um valor

```
DECLARE @nome <tipo>[(tam)] = <value>
```

Permite alterar o valor de uma variável

```
SET @nome = <value>
```

Permite alterar o valor de N variáveis

```
SELECT @nome1 = <value1>, @nome1 = <value1>
```

Veja Alguns Exemplos Práticos

USANDO SET:

```
DECLARE @idade INT = 30;  
PRINT 'Valor de @idade = ';  
PRINT @idade;  
SET @idade = 18;  
PRINT 'Valor de @idade = ';  
PRINT @idade;
```

SET: Define um valor por vez!

USANDO SELECT:

```
DECLARE @nome VARCHAR(30), @codigo INT;  
SELECT @nome = 'Fernando', @codigo = 1;  
PRINT 'Valor de @nome = ';  
PRINT @nome;  
PRINT 'Valor de @codigo = ';  
PRINT @codigo;
```

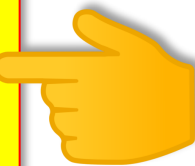
SELECT: Define múltiplos valores!

VALORES DE UM REGISTRO:

```
DECLARE @nome NVARCHAR(30), @cidade NVARCHAR(30);  
SELECT @nome=nome, @cidade=cidade FROM Funcionario WHERE idFunc=1;  
PRINT 'Valor de @nome = ';  
PRINT @nome;  
PRINT 'Valor de @cidade = ';  
PRINT @cidade;
```

Obtém valor de um único registro!

Se trazer mais de um registro, o último é o que será adotado!



Uso de Expressões e Funções com Variáveis

- ❑ É possível usar **qualquer expressão e função com variáveis**;
- ❑ Em outras palavras, **os seu valores são atribuídos a elas**;
- ❑ *Imagine salvar o número de registros da tabela para uso posterior;*
- ❑ *Ou concatenar o nome de um arquivo com o seu respectivo diretório.*

Formas diferentes de concatenar valores!

```
DECLARE @msg VARCHAR(50);  
SET @msg = 'Olá, '  
SET @msg += 'Mundo!';  
PRINT 'Valor do Texto = ' + @msg;
```

```
DECLARE @numreg INT;  
SELECT @numreg=COUNT(*) from  
Funcionario;  
PRINT 'Número de Registros = ' +  
CAST(@numreg AS VARCHAR);
```

Apresenta o número de registros!

Where e Having com Variáveis

- ❑ É comum usarmos **valores literais** para **compor as expressões**;
- ❑ Porém, algumas vezes **não conhecemos previamente esses valores**;
- ❑ Nesses casos é indicado **o uso de variáveis** em seu lugar.

Busca os nomes que começam com 'pe'!

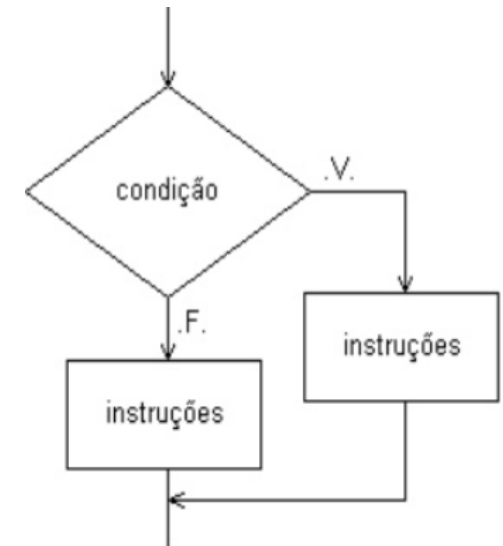
```
DECLARE @nm NVARCHAR(30);  
SET @nm = N'pe%';  
SELECT * FROM Funcionario  
WHERE nome LIKE @nm;
```

```
DECLARE @val MONEY = 1500;  
SELECT cidade, SUM(salario) FROM  
Funcionario GROUP BY cidade  
HAVING SUM(salario) > @val;
```

Mostra apenas as cidades cujo agrupamento do salário é maior do que 1500.

Estrutura de Seleção IF...ELSE

- ❑ A **estrutura IF...ELSE** é usada de forma similar as linguagens de programação tradicionais;
- ❑ A sua adoção permite **definir os fluxos possíveis do código** dentro do seu script T-SQL;
- ❑ As declarações são executadas mediante a **avaliação de uma expressão condicional**;
- ❑ Se TRUE, o bloco do IF é executado. Agora se FALSE, o bloco executado é o do ELSE.



Sintaxe do IF...ELSE

Linha Única:

```
IF <condição> <declaração>  
ELSE <declaração>
```

```
IF <condição>  
    <declaração>  
ELSE  
    <declaração>
```

Linhas Múltiplas:

```
IF <condição> BEGIN  
    <declaração 1>  
    <declaração 2>  
    <declaração N>  
END  
ELSE BEGIN  
    <declaração 1>  
    <declaração 2>  
END
```

BEGIN E END:

*Designam as declarações
que fazem parte do bloco
de comandos.*



Exemplos do Uso de IF...ELSE

Verifica a quantidade de registros!

```
DECLARE @qtd INT;
SELECT @qtd=COUNT(*) FROM Funcionario;
IF(@qtd > 10)
    PRINT 'Há mais do que dez registros!';
ELSE
    PRINT 'Há menos do que dez registros!';
```

```
DECLARE @nm NVARCHAR(30);
SELECT @nm=nome FROM Funcionario WHERE idFunc=3;
IF CHARINDEX('ben',@nm) > 0 BEGIN
    PRINT 'O nome contém "ben".';
    PRINT 'O nome é = ' + @nm;
END;
ELSE BEGIN
    PRINT 'O nome não contém "ben".';
    PRINT 'O nome é = ' + @nm;
END;
```

Verifica se o texto contém uma string!

Os IF's podem estar aninhados!

```
DECLARE @n1 real, @n2 real, @md real;
SET @n1 = 9.2;
SET @n2 = 7.8;
IF (@n1 < 0) OR (@n1 > 10)
    PRINT 'NOTA N1 INVÁLIDA!';
ELSE BEGIN
    IF (@n2 < 0) OR (@n2 > 10)
        PRINT 'NOTA N2 INVÁLIDA!';
    ELSE BEGIN
        SET @md=(@n1+@n2)/2;
        PRINT 'MÉDIA = ' + CAST(@md as VARCHAR);
    END;
END;
```

As Palavras-Chave BEGIN e END

- ❑ A T-SQL usa essas palavras para **agrupar as declarações em blocos**;
- ❑ Elas não alteram a **ordem de execução** das declarações;
- ❑ Também não servem para **limitar o escopo** das variáveis.

Código em C#:

Não compila, pois está fora do escopo.

```
class Exemplo
{
    public static void Main()
    {
        int j = 10;
    }

    System.Console.WriteLine(j);
}
```

Código em T-SQL:

Compila, pois o escopo não é delimitado.

```
BEGIN
    DECLARE @j INT = 10;
END;
PRINT @j;
```

Usando IF...EXISTS

- ❑ **Checa o resultado de um SELECT** antes de executar as declarações do IF;
- ❑ O resultado dessa função sempre será **TRUE** ou **FALSE**;
- ❑ *Um exemplo é ver se uma tabela existe antes de dropá-la.*

```
IF [NOT] EXISTS (SELECT * FROM <TABLE> [WHERE <condição>]) BEGIN  
    <declaração 1>  
    <declaração 2>  
    <declaração N>  
END
```

Veja Alguns Exemplos Práticos

Checa se não há funcionário com o nome terminado em “dro”

```
DECLARE @nm NVARCHAR(20)='%dro';  
IF NOT EXISTS(SELECT * FROM Funcionario WHERE nome like @nm) BEGIN  
    PRINT 'Não há ninguém com o nome terminado em: ' + @nm;  
END;
```

Verifica se existe o funcionário com ID 4

Nesses exemplos o uso de asterisco (*) é perfeitamente aceitável, pois retorna poucos registros.

```
DECLARE @id INT = 4;  
IF EXISTS(SELECT * FROM Funcionario WHERE idFunc=@id) BEGIN  
    PRINT 'Existe um Funcionário com esse ID!';  
END;  
ELSE BEGIN  
    PRINT 'Não existe um funcionário com esse ID!';  
END;
```



Estrutura CASE

- ❑ Compara um **dado em relação a outros** valores (opções);
- ❑ Evita a **repetição excessiva de IF's** dentro do código;
- ❑ Fornece um **melhor entendimento e organização** da lógica.

```
CASE <dado>  
  WHEN <opção 1> THEN <retorno>  
  WHEN <opção 2> THEN <retorno>  
  WHEN <opção 3> THEN <retorno>  
  ELSE <retorno>  
END
```

Exemplo de Estrutura CASE

Retorna um inteiro
referente ao dia da semana

```
SELECT nome, datanasccto,  
       CASE DATEPART(WEEKDAY, datanasccto)  
         WHEN 1 THEN 'DOMINGO'  
         WHEN 2 THEN 'SEGUNDA'  
         WHEN 3 THEN 'TERÇA'  
         WHEN 4 THEN 'QUARTA'  
         WHEN 5 THEN 'QUINTA'  
         WHEN 6 THEN 'SEXTA'  
         WHEN 7 THEN 'SABÁDO'  
       END AS Dia_Semana  
FROM Funcionario;
```

	nome	datanasccto	Dia_Semana
1	pedro	1990-08-28	TERÇA
2	pedro	1938-06-12	DOMINGO
3	rubens	1938-06-12	DOMINGO

Estrutura de Repetição WHILE

- ❑ Possibilita executar um código **determinado número de vezes** (iteração);
- ❑ Usa uma **expressão condicional** para definir o **ponto de parada**;
- ❑ Um **contador** é usado em conjunto com a expressão condicional;
- ❑ A cada loop o **contador é incrementado ou decrementado**.

```
WHILE <condição> BEGIN  
    <declaração 1>  
    <declaração 2>  
    [<declaração N>]  
END;
```


Exemplos do Uso do WHILE

```
DECLARE @x INT = 1;  
WHILE (@x <= 10) BEGIN  
    PRINT @x;  
    SET @x+=1;  
END;
```

Imprime os
valores de 1 a 10!

Atualizar campo por campo é muito
ineficiente devido a quantidade de “*locks*” e
commits realizados nas tabelas. Por esse
motivo é importante planejar corretamente
os seus *batches*.

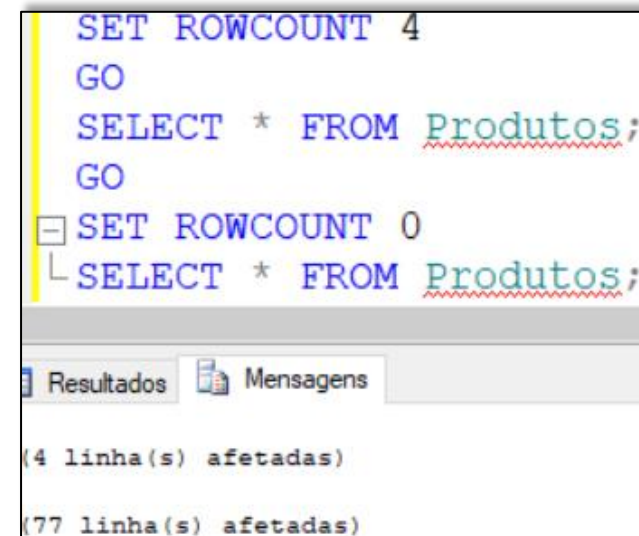


```
DECLARE @id INT = 1;  
WHILE EXISTS(SELECT * FROM Funcionario WHERE salario IS NOT NULL) BEGIN  
    UPDATE Funcionario SET salario=NULL WHERE idFunc=@id;  
    SET @id+=1;  
END;  
PRINT 'Atualização Realizada com Sucesso!';
```

Atualiza o campo
salário para NULO!

Sobre o ROWCOUNT

- ❑ Ao executar uma declaração de *insert*, *update* ou *delete* vários campos podem ser afetados;
- ❑ Para não onerar o processamento pode-se **informar o número de campos afetados por vez**;
- ❑ Os **registros são divididos em lotes** para serem manipulados;
- ❑ Essa técnica é muito usada em **conjunto com a estrutura While**.



```
SET ROWCOUNT 4
GO
SELECT * FROM Produtos;
GO
SET ROWCOUNT 0
SELECT * FROM Produtos;
```

Resultados Mensagens

(4 linha(s) afetadas)

(77 linha(s) afetadas)

Script usando o ROWCOUNT

(Parte 1)

```
IF EXISTS (SELECT * FROM sys.objects WHERE  
            object_id = OBJECT_ID(N'Produtos') AND TYPE IN (N'U'))  
    DROP TABLE Produtos;  
  
GO
```

Script usando o ROWCOUNT

(Parte 2)

```
CREATE TABLE Produtos (idProd int not null identity primary key,  
                           nome varchar(100),  
                           preco money)
```

```
GO
```

Script usando o ROWCOUNT

(Parte 3)

```
SET ROWCOUNT 0;
```

```
INSERT INTO Produtos(nome,preco)  
SELECT ProductName,UnitPrice from Products;  
PRINT 'Tabela Copiada!';
```

```
GO
```

Script usando o ROWCOUNT

(Parte 4)

```
SET ROWCOUNT 10;

WHILE EXISTS(SELECT * FROM Produtos WHERE preco != 0) BEGIN
    UPDATE Produtos SET preco = 0 WHERE preco != 0;
    PRINT 'ATUALIZANDO 10 REGISTROS POR VEZ!';
END;
PRINT 'ATUALIZAÇÃO CONCLUÍDA!';
```

Script usando o ROWCOUNT

(Resultado)

(77 linha(s) afetadas)
Tabela Copiada!

(10 linha(s) afetadas)
ATUALIZANDO 10 REGISTROS POR VEZ!

(10 linha(s) afetadas)
ATUALIZANDO 10 REGISTROS POR VEZ!

(10 linha(s) afetadas)
ATUALIZANDO 10 REGISTROS POR VEZ!

(10 linha(s) afetadas)
ATUALIZANDO 10 REGISTROS POR VEZ!

(10 linha(s) afetadas)
ATUALIZANDO 10 REGISTROS POR VEZ!

(10 linha(s) afetadas)
ATUALIZANDO 10 REGISTROS POR VEZ!

(10 linha(s) afetadas)
ATUALIZANDO 10 REGISTROS POR VEZ!

(7 linha(s) afetadas)
ATUALIZANDO 10 REGISTROS POR VEZ!
ATUALIZAÇÃO CONCLUÍDA!

Note que as
atualizações
foram feitas em
etapas!

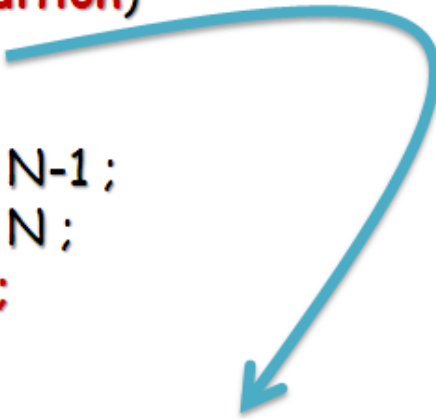
Saindo de um Loop

- ❑ Na maioria das vezes o **WHILE** é **executado** até que a **expressão condicional retorne falso**;
- ❑ Em alguns casos podemos **solicitar o encerramento** do loop através da palavra **BREAK**;
- ❑ Para isso geralmente temos um **IF** **dentro da estrutura de repetição** para controlar a parada.

```
while (condition)
{
    Statement 1 ;
    Statement 2 ;
    Statement 3 ;
    .....
    .....
    if ( If Condition )
        break;

    Statement N-1 ;
    Statement N ;
    Increment;
}

OutsideStatement 1;
```



Exemplo de Break na Repetição

```
DECLARE @i INT = 1;  
WHILE (@i <= 100) BEGIN  
    PRINT @i;  
    IF (@i = 20) BEGIN  
        PRINT 'SAINDO DO LOOP!';  
        BREAK;  
    END;  
    SET @i+=1;  
END;
```

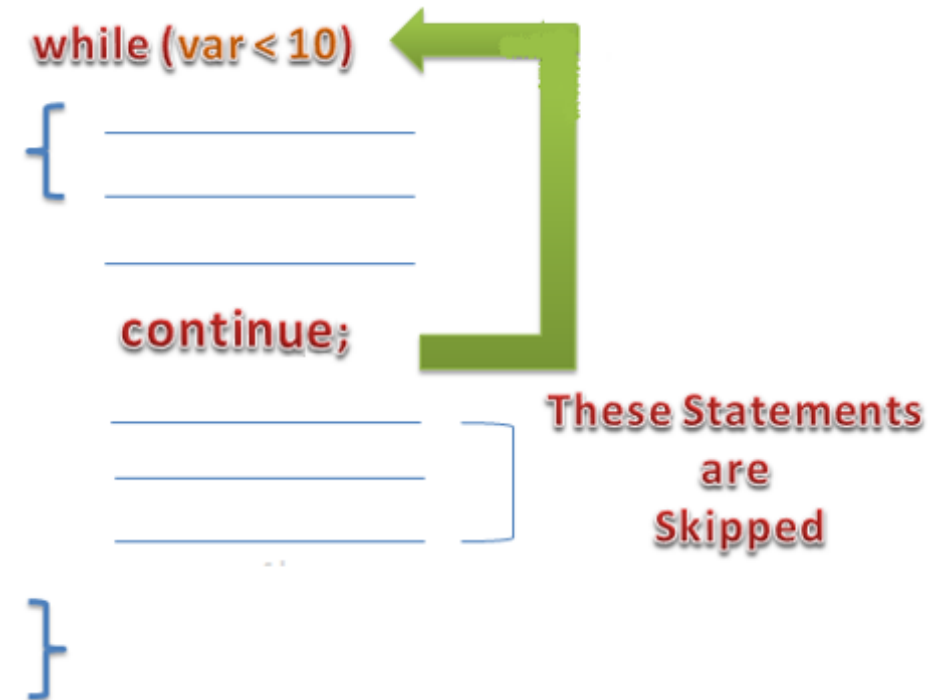
Quando @i = 20 o loop
é encerrado!

RESULTADO

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
SAINDO DO LOOP!
```

Comando CONTINUE

- ❑ O comando **CONTINUE** força o loop a **voltar no início** no momento em que for encontrado;
- ❑ Em outras palavras, o código **localizado após o comando CONTINUE não será executado**;
- ❑ A sua utilização, *assim como o BREAK*, geralmente ocorre através de um **IF aninhado**.



Usando o Comando CONTINUE

```
DECLARE @x INT = 0;  
WHILE (@x < 10) BEGIN  
    SET @x+=1;  
    IF (@x = 5)  
        CONTINUE;  
    PRINT @x;  
END;
```

O **valor 5** não é
impresso!

RESULTADO

1
2
3
4
6
7
8
9
10

Mãos à Obra!!!



Envie por e-mail conforme o padrão apresentado na Aula 1