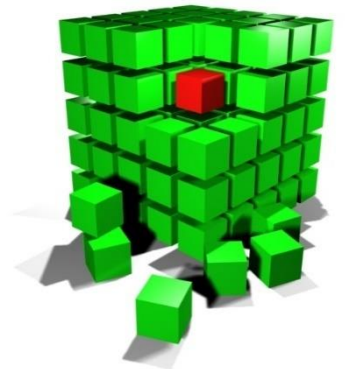
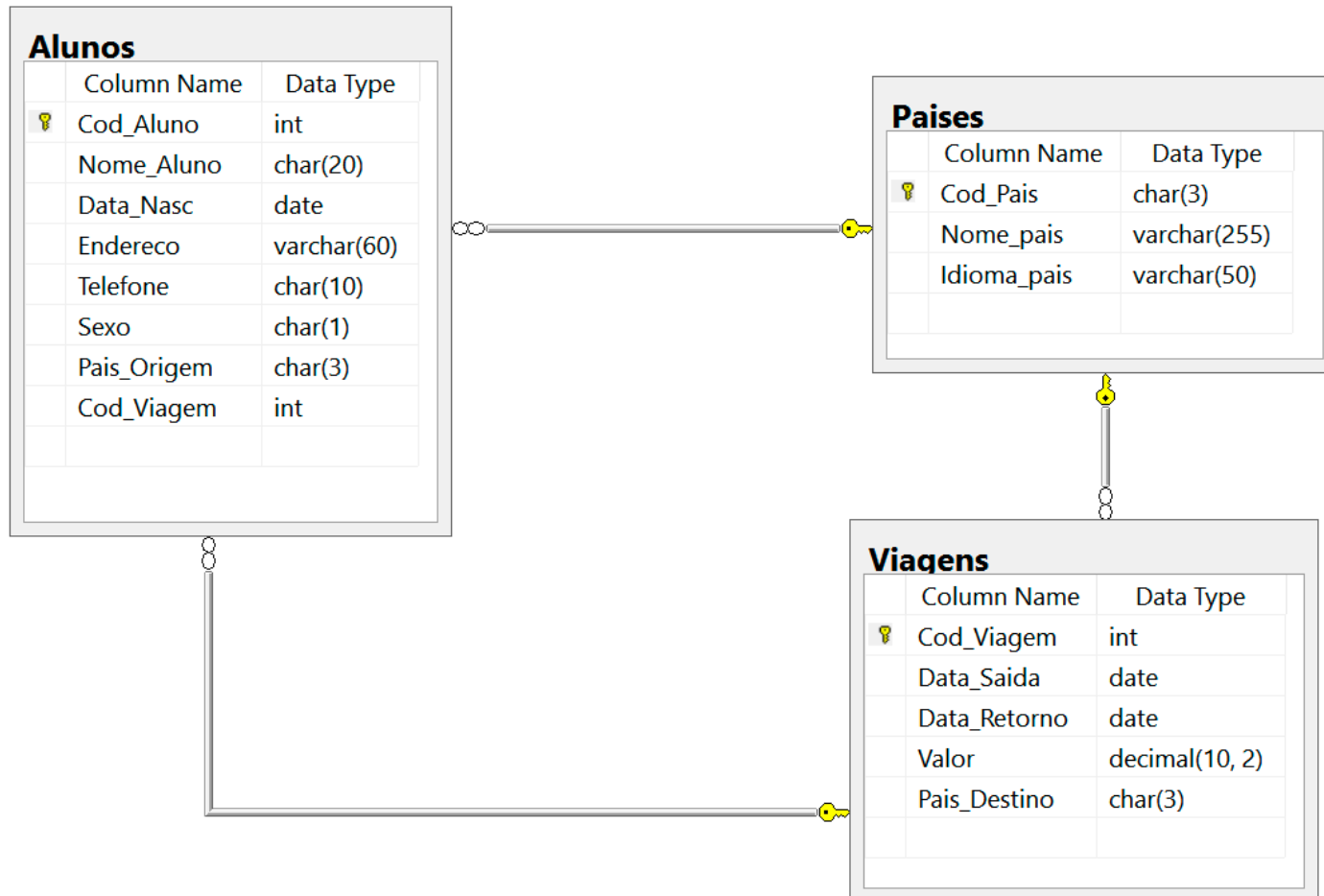


- **Aula 015 – Linguagem SQL**
 - Sequências.
 - Colunas do tipo IDENTITY.
 - SQL Procedural.



■ Banco de dados INTERCAMBIO



■ Sequências

- Uma **sequência**, ou **SEQUENCE**, é um objeto associado a um esquema definido pelo usuário que gera uma sequência de valores numéricos de acordo com a especificação com a qual a sequência foi criada.
- A sequência de valores numéricos é gerada em ordem crescente ou decrescente, dentro de um intervalo definido, e pode ser configurada para reiniciar quando esse intervalo se esgotar.



■ Sequências

- As sequências, ao contrário de **colunas de identidade**, ou **IDENTITY**, não são associadas a tabelas específicas. Os aplicativos fazem referência a um objeto de sequência para recuperar seu próximo valor. A relação entre sequências e tabelas é controlada pelo aplicativo.
- Os aplicativos de usuário podem referenciar um objeto de sequência e coordenar os valores nas várias linhas e tabelas.



■ Sequências

- A diferença em relação ao **IDENTITY** se dá pelo fato de que uma **SEQUENCE** pode ser acionada sempre que necessária, independente de tabelas e campos do banco de dados.
- Entretanto, em relação à funcionalidade, a sequência mantém semelhanças com o **IDENTITY**.
- No **Access**, podemos utilizar o **AutoNumber**, para definir uma coluna de tabela preenchida automaticamente com valores numéricos exclusivos.



- **Sequências no ORACLE e no SQL SERVER**
 - São objetos independentes no banco de dados.
 - Possuem um nome e podem ser utilizadas em qualquer posição de comando.
 - Não são atreladas a uma tabela ou coluna.
 - Geram um valor numérico automático que pode ser atribuído a qualquer coluna de qualquer tabela
 - O atributo de tabela ao qual é atribuído um valor com base em uma sequência pode ser editado e modificado.
 - É possível criar e excluir uma sequência a qualquer momento.



■ Sintaxe do SQL SERVER

```
CREATE SEQUENCE [schema_name.] sequence_name  
    [AS [tipo_de_dados_interno|  
tipo_de_dados_definido_pelo_usuario]]  
    [START WITH <valor_inicial>]  
    [INCREMENT BY <valor_incremento>]  
    [{MINVALUE [<valor_mínimo>]} | {NO MINVALUE}]  
    [{MAXVALUE [<valor_máximo>]} | {NO MAXVALUE}]  
    [CYCLE | {NO CYCLE }]  
    [{CACHE [<tamanho_cache>]} | {NO CACHE}]  
[;]
```



■ Diferenças entre SEQUENCE e IDENTITY

SEQUENCE

É um objeto independente, que pode ser utilizado pra preencher qualquer coluna (inclusive, mais de uma na mesma tabela) do tipo **INT**, **BIGINT**, **SMALLINT**, **TINYINT**, **DECIMAL** com escala 0 ou **NUMERIC** com escala 0, de uma ou mais tabelas.

É populada quando for chamada. Ou seja, pode ser populada a cada inserção ou apenas quando alguma condição for atendida.

Ao ser iniciada em uma tabela já populada, os registros anteriores não serão alterados.

Deve ser chamado manualmente para gerar o sequencial.

Possui permissões à parte.

IDENTITY

É associado a uma coluna de uma tabela.

É populada em cada inserção.

Ao ser iniciada na tabela, a coluna inteira é populada com o sequencial.

O sequencial é gerado automaticamente.

Não requer permissões adicionais além da tabela.

■ Diferenças entre SEQUENCE e IDENTITY

SEQUENCE

Pode ser definido valor mínimo e máximo (Ex: 1 a 100).

O sequencial pode ser reiniciado automaticamente ao atingir o valor máximo (parâmetro CYCLE).

Pode ser gerado um novo sequencial em comandos de **UPDATE**, caso necessário.

Disponível a partir do SQL Server 2012.

O valor atual do sequencial pode ser consultado através da **VIEW SYS.sequences**.

IDENTITY

O valor máximo é o limite do tipo de dado da coluna.

Ao atingir o valor máximo não é possível inserir mais registros.

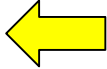
O sequencial é gerado apenas no **INSERT** dos dados.

Disponível a partir do SQL Server 6.0 (SQL 95).

O valor atual do sequencial pode ser obtido através da **VIEW SYS.identity_columns**.



■ Utilizando o IDENTITY no SQL SERVER

```
CREATE TABLE Alunos (  
    Cod_Aluno    INT IDENTITY PRIMARY KEY,   
    Nome_Aluno   CHAR(20)          NOT NULL,  
    Data_Nasc    DATE              NOT NULL,  
    Endereco     VARCHAR(60)       NOT NULL,  
    Telefone     CHAR(10)          NOT NULL,  
    Sexo         CHAR(1),  
    Pais_Origem  CHAR(3)          NOT NULL FOREIGN KEY  
    REFERENCES Pais(Cod_Pais),  
    Cod_Viagem   INT              NOT NULL FOREIGN KEY  
    REFERENCES Viagens(Cod_Viagem)  
)
```



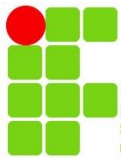
■ Utilizando o IDENTITY no SQL SERVER

-- Tabela AERONAVES

```
CREATE TABLE AERONAVES (  
    Cod_Aeronave INT IDENTITY PRIMARY KEY,  
    Modelo VARCHAR(50) NOT NULL  
)  
GO
```

O valor de Cod_Aeronave começa em 1,
e é incrementado de 1 em 1



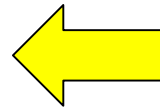


■ Utilizando o IDENTITY no SQL SERVER

-- Insere algumas aeronaves

INSERT INTO AERONAVES VALUES

('Boeing 707'),
('Boeing 737'),
('Boeing 747'),
('Embraer ERJ-145'),
('Vickers VC-10')



Durante a inserção, não fornecemos o valor de um atributo do tipo IDENTITY

GO

-- Exibe os dados das aeronaves

SELECT * FROM AERONAVES

GO



■ Utilizando o IDENTITY no SQL SERVER

Cod_Aeronave	Modelo
1	Boeing 707
2	Boeing 737
3	Boeing 747
4	Embraer ERJ-145
5	Vickers VC-10



(5 row(s) affected)



■ Utilizando o IDENTITY no SQL SERVER

- Tenta inserir uma nova aeronave,
- fornecendo o valor de um campo IDENTITY

```
INSERT INTO AERONAVES (Cod_Aeronave, Modelo) VALUES  
    (6, 'Airbus A300')  
GO
```



Msg 544, Level 16, State 1, Line 754

Cannot insert explicit value for identity column in
table 'AERONAVES' when IDENTITY_INSERT is set to OFF.



■ Utilizando o IDENTITY no SQL SERVER

-- Permite inserir valores explícitos na coluna de identidade de uma tabela

```
SET IDENTITY_INSERT AERONAVES ON
```

GO

-- Tenta inserir uma nova aeronave, fornecendo o valor de um campo IDENTITY

```
INSERT INTO AERONAVES (Cod_Aeronave, Modelo) VALUES  
(6, 'Airbus A300')
```

GO

-- Exibe os dados das aeronaves

```
SELECT * FROM AERONAVES
```

GO



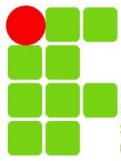
■ Utilizando o IDENTITY no SQL SERVER

Cod_Aeronave Modelo

```
-----  
1                Boeing 707  
2                Boeing 737  
3                Boeing 747  
4                Embraer ERJ-145  
5                Vickers VC-10  
6                Airbus A300
```

(6 row(s) affected)





■ Utilizando o IDENTITY no SQL SERVER

-- Altera o valor de IDENTITY_INSERT

SET IDENTITY_INSERT AERONAVES OFF ←

GO

-- Tabela VEICULOS, com um IDENTITY com incremento 10

CREATE TABLE VEICULOS (

Codigo INT IDENTITY(1, 10) PRIMARY KEY, ←

Modelo VARCHAR(50) NOT NULL

)

GO



■ Utilizando o IDENTITY no SQL SERVER

-- Insere alguns veículos

```
INSERT INTO VEICULOS VALUES
```

```
    ( 'Ferrari' ),
```

```
    ( 'Camaro' ),
```

```
    ( 'Fusca' )
```

```
GO
```

-- Exibe os dados

```
SELECT * FROM VEICULOS
```

```
GO
```



■ Utilizando o IDENTITY no SQL SERVER

-- Exibe o valor do incremento utilizado

```
SELECT IDENT_INCR('AERONAVES') AS 'Inc. AERONAVES',  
       IDENT_INCR('VEICULOS')  AS 'Inc. VEICULOS'
```

GO

-- Exibindo o último valor de IDENTITY que foi utilizado

```
SELECT @@IDENTITY AS 'Último IDENTITY',  
       IDENT_CURRENT('VEICULOS')  AS 'IDENTITY -  
VEICULOS',  
       IDENT_CURRENT('AERONAVES') AS 'IDENTITY -  
AERONAVES'
```

GO

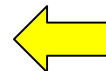
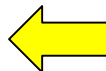


■ Utilizando o IDENTITY no SQL SERVER

Codigo	Modelo
1	Ferrari
11	Camaro
21	Fusca

Inc. AERONAVES	Inc. VEICULOS
1	10

Último IDENTITY	IDENTITY - VEICULOS	IDENTITY - AERONAVES
21	21	6





■ Utilizando SEQUENCES no SQL SERVER

-- Cria uma sequência que começa em 1, com incremento 1

```
CREATE SEQUENCE Incrementa1 AS INT
```

```
START WITH 1
```

```
INCREMENT BY 1
```

```
GO
```

-- Cria uma sequência que começa em 10, com incremento 100

```
CREATE SEQUENCE Incrementa100 AS INT
```

```
START WITH 10
```

```
INCREMENT BY 100
```

Nome das sequências

```
GO
```



■ Utilizando SEQUENCES no SQL SERVER

- Cria uma sequência que começa em
- 1000, com incremento -100

```
CREATE SEQUENCE Incrementa1000 AS INT  
    START WITH 1000  
    INCREMENT BY -100  
  
GO
```



■ Utilizando SEQUENCES no SQL SERVER

-- Exibe informações sobre as sequências

```
SELECT name AS 'Nome',  
        create_date AS 'Data de Criação',  
        start_value AS 'Valor Inicial',  
        incremente AS 'Incremento',  
        minimum_value AS 'Valor Mínimo',  
        maximum_value AS 'Valor Máximo',  
        current_value AS 'Valor Atual'  
  
FROM SYS.sequences  
  
GO
```



■ Utilizando SEQUENCES no SQL SERVER

Nome	Data de Criação	Valor Inicial
-----	-----	-----
Incremental1	2018-08-27 13:41:20.570	1
Incremental100	2018-08-27 13:43:39.967	10
Incremental1000	2018-08-27 13:46:11.283	1000

Incremento	Valor Mínimo	Valor Máximo	Valor Atual
-----	-----	-----	-----
1	-2147483648	2147483647	1
100	-2147483648	2147483647	10
-100	-2147483648	2147483647	1000



■ Utilizando SEQUENCES no SQL SERVER

-- Criando uma sequência com mais parâmetros

```
CREATE SEQUENCE IncrementaDecimal AS DECIMAL(3,0)
```

```
START WITH 125
```

```
INCREMENT BY 25
```

```
MINVALUE 100
```

```
MAXVALUE 200
```

```
CYCLE
```

```
CACHE 3
```

```
GO
```



■ Utilizando SEQUENCES no SQL SERVER

-- Exibe o primeiro valor das sequências

```
SELECT NEXT VALUE FOR Incrementa1      AS 'Incrementa1',  
       NEXT VALUE FOR Incrementa100   AS 'Incrementa100',  
       NEXT VALUE FOR Incrementa1000 AS 'Incrementa1000',  
       NEXT VALUE FOR IncrementaDecimal AS  
'IncrementaDecimal'  
GO
```

Incrementa1	Incrementa100	Incrementa1000	IncrementaDecimal
1	10	1000	125



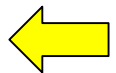
■ Utilizando SEQUENCES no SQL SERVER

- Cria uma tabela temporária, para ilustrar a
- utilização das sequências

```
CREATE TABLE #TestaSequencia (  
    ID    INT,  
    Nome  CHAR(20)  
)  
GO
```

- Reiniciando o valor de uma sequência

```
ALTER SEQUENCE Incrementa100 RESTART WITH 10  
GO
```



■ Utilizando SEQUENCES no SQL SERVER

```
-- Utiliza a sequência INCREMENTA100 para  
-- inserir valores na tabela #TestaSequencia
```

```
INSERT INTO #TestaSequencia (ID, Nome) VALUES  
    (NEXT VALUE FOR Incrementa100, 'Ana'),  
    (NEXT VALUE FOR Incrementa100, 'Maria'),  
    (NEXT VALUE FOR Incrementa100, 'João')
```

```
GO
```

```
-- Exibe os valores
```

```
SELECT * FROM #TestaSequencia
```

```
GO
```



■ Utilizando SEQUENCES no SQL SERVER

ID	Nome
-----	-----
10	Ana
110	Maria
210	João

(3 row(s) affected)



■ Utilizando SEQUENCES no SQL SERVER

-- Recuperando o valor atual de uma sequência

```
SELECT current_value AS 'Valor Atual'
```

```
FROM SYS.sequences
```

```
WHERE name = 'Incrementa100'
```

```
GO
```

Valor Atual

210 ←



■ Utilizando SEQUENCES no SQL SERVER

-- Remove uma sequência

```
DROP SEQUENCE Incrementa100
```

```
GO
```

-- Tenta inserir um registro utilizando a sequência

-- INCREMENTA100

```
INSERT INTO #TestaSequencia (ID, Nome)
```

```
VALUES
```

```
(NEXT VALUE FOR Incrementa100, 'José')
```

```
GO
```



■ SQL procedural

- A definição padrão da linguagem SQL não fornece suporte à programação estruturada, tal qual é normalmente suportado pelas linguagens de programação.
- Dessa forma, a execução de estruturas condicionais e de estruturas de repetição ficam a cargo dos chamados **dialetos SQL**, implementados por cada desenvolvedor de um SGBD específico.



■ SQL procedural

- Um dialeto SQL funciona como uma extensão da linguagem SQL, que permite, dentre outras coisas, a utilização de estruturas de programação procedural de controle de fluxo (**IF-THEN-ELSE**, **WHILE**, etc.) para representação de lógica, a declaração e designação de variáveis, a declaração e utilização de procedimentos e funções, e o gerenciamento de erros.



- **Módulo de armazenamento persistente**
 - Para resolver a falta de funcionalidade procedural na linguagem SQL e fornecer alguma padronização dentre as várias ofertas dos fornecedores, o padrão SQL-99 definiu a utilização de módulos permanentemente armazenados.
 - O **módulo de armazenamento persistente** (MAP) é um bloco de código contendo comandos-padrão e extensões procedurais de SQL, o qual é armazenado e executado diretamente no servidor do SGBD. A implementação do suporte ao MAP é deixada a cargo de cada fornecedor de um SGBD.



■ Dialetos SQL

- **Microsoft SQL Server** – implementa o MAP por meio do dialeto **T-SQL**, e de outras extensões da linguagem, como as que utilizam os recursos da família .NET.
- **Oracle** – implementa o MAP por meio de sua linguagem procedural denominada de PL/SQL.
- **PostgreSQL** – implementa o MAP por meio do dialeto PL/pgSQL.
- **MySQL** – implementa o MAP por meio do dialeto MySQL Dialect.
- **Microsoft Access** – implementa o MAP por meio do dialeto Jet SQL.



■ Dialetos SQL

- **T-SQL** – extensão da linguagem SQL lançada pela Microsoft para ser usada pelo SQL Server. Tem como características as variações de suporte ao processamento de strings, datas, funções, etc. Tem modo ágil de declaração de comandos DML, principalmente **UPDATE** e **DELETE** e apresenta **diversas variáveis locais**.
- **PL/SQL** – é a extensão da linguagem SQL lançada pela SUN para ser usada pelo Oracle. É a linguagem que tem como principal característica trabalhar com seus comandos como blocos, sendo processados individualmente. Tem como estrutura básica o seguinte modo: **DECLARE** → **SELECTION** → **BEGIN** → **EXCEPTION** → **END**.



- **SQL procedural**
 - Declaração de variáveis.
 - Estruturas de decisão.
 - Estruturas de repetição.
 - Procedimentos armazenados.
 - Funções.
 - Gatilhos.
 - Cursores.



■ Declaração de variáveis em T-SQL

- As variáveis são declaradas no corpo de um lote ou procedimento com a instrução **DECLARE**, e os valores são atribuídos com uma instrução **SET** ou **SELECT**.
- As **variáveis de cursor** podem ser declaradas com essa instrução e usadas com outras instruções relacionadas ao cursor.
- Depois da declaração, todas as variáveis são inicializadas como **NULL**, a menos que um valor seja fornecido como parte da declaração.



■ Declaração de variáveis em T-SQL

-- Declara uma variável

DECLARE @nome AS VARCHAR(100) ←

-- Atribui um valor

SET @nome = 'Carlos Pereira' ←

-- Utiliza a variável em uma consulta SQL

```
SELECT Cod_Aluno AS 'Código',  
       Nome_Aluno AS 'Nome do Aluno',  
       Endereco AS 'Endereço'
```

FROM Alunos

WHERE Nome_Aluno LIKE @nome ←

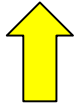
GO



■ Declaração de variáveis em T-SQL

-- Atribui um valor para uma variável durante a sua declaração

```
DECLARE @nome AS VARCHAR(100) = 'Carlos Pereira'
```



-- Utiliza a variável em uma consulta SQL

```
SELECT Cod_Aluno AS 'Código',  
       Nome_Aluno AS 'Nome do Aluno',  
       Endereco AS 'Endereço'  
FROM Alunos  
WHERE Nome_Aluno LIKE @nome  
GO
```




■ Declaração de variáveis em T-SQL

-- Atribui um valor para uma variável
-- utilizando um SELECT

```
DECLARE @rows AS INT
```

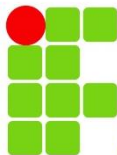
```
SET @rows = (SELECT COUNT(*) FROM  
Viagens)
```



```
SELECT @rows AS 'Total de Viagens'
```

```
GO
```





■ Estruturas de decisão em T-SQL

- O T-SQL permite a utilização de estruturas de decisão do tipo **IF-ELSE** e **SELECT CASE-WHEN-THEN**.
- Estruturas do tipo **IF-ELSE** impõe condições na execução de uma instrução T-SQL. A instrução T-SQL que segue uma palavra-chave **IF** e sua condição será executada se a condição for satisfeita. A palavra-chave opcional **ELSE** introduz outra instrução T-SQL que será executada quando a condição **IF** não for atendida.



■ Exemplo IF-ELSE

-- Declara as variáveis

```
DECLARE @A AS INT = 10,  
        @B AS INT = 100,  
        @maior AS INT
```

-- Verifica qual é o maior valor

```
IF @A > @B  
    SET @maior = @A  
ELSE  
    SET @maior = @B
```



■ Exemplo IF-ELSE

-- Exibe o maior valor

```
PRINT 'O maior valor é: ' + CAST(@maior AS  
VARCHAR)
```

```
GO
```

O maior valor é: 100 ←



■ Exemplo IF-ELSE

-- Trecho que verifica se um número é par

```
DECLARE @numero AS INT = 240
```

```
IF ((@numero % 2) = 0) ←
```

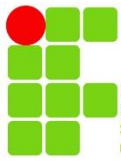
```
    PRINT '0 número ' + CAST(@numero AS  
    VARCHAR) + ' é par!'
```

```
ELSE
```

```
    PRINT '0 número ' + CAST(@numero AS  
    VARCHAR) + ' é ímpar!'
```

```
GO
```





■ Exemplo IF-ELSE

-- Consulta para exibir se um aluno é homem ou mulher

```
SELECT Cod_Aluno    AS 'Código',  
       Nome_Aluno   AS 'Nome do Aluno',  
       Endereco     AS 'Endereço',  
       Sexo,  
       IF Sexo = 'F'  
         SELECT 'Mulher'  
       ELSE  
         SELECT 'Homem'  
       END AS 'Homem | Mulher'  
FROM Alunos
```

← Não funciona!



■ Estrutura CASE

- A instrução **CASE** avalia uma lista de condições e retorna uma das várias expressões de resultado possíveis. Ela possui dois formatos, os quais oferecem suporte para um argumento **ELSE** opcional.
- A expressão **CASE** simples compara uma expressão com um conjunto de expressões simples para determinar o resultado.
- A expressão **CASE** pesquisada avalia um conjunto de expressões booleanas para determinar o resultado.



■ Estrutura CASE

- A instrução **CASE** pode ser usada em qualquer instrução ou cláusula que permita uma expressão válida, como por exemplo, **SELECT**, **UPDATE**, **DELETE** e **SET**, e em cláusulas, como **IN**, **WHERE**, **ORDER BY** e **HAVING**.



■ Estrutura CASE

-- Expressão CASE simples

```
CASE expressão_de_entrada ←  
    WHEN condição THEN resultado  
    [...n]  
    [ELSE resultado_ELSE]  
END
```

Sintaxes Possíveis

-- Expressão CASE pesquisada

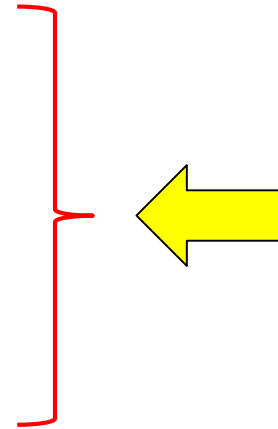
```
CASE  
    WHEN expressão_booleana THEN resultado ←  
    [...n]  
    [ELSE resultado_ELSE]  
END
```



■ Exemplo CASE

-- Consulta para exibir se um aluno é homem ou mulher

```
SELECT Cod_Aluno AS 'Código',  
       Nome_Aluno AS 'Nome do Aluno',  
       Endereco AS 'Endereço',  
       Sexo,  
       CASE Sexo  
         WHEN 'M' THEN 'Homem'  
         WHEN 'F' THEN 'Mulher'  
         ELSE 'Não declarado'  
       END AS 'Tipo de Sexo'  
FROM Alunos
```



■ Exemplo CASE

Código	Nome do Aluno	Endereço	Sexo	Tipo de Sexo

1	Maria Cristina	Rua João XXIII, 15 - São Paulo	F	Mulher
2	Ana Paula Lima	Rua Mauro Silva, 1908 - São Paulo	F	Mulher
3	Carlos Renato	Av. Faria Lima, 347 - São Paulo	M	Homem
4	Hugo Silva	Av. da Consolação, 1216 - São Paulo	M	Homem
5	Marcos Antônio	Rua Agripino Lopes, 100 - São Paulo	M	Homem
6	Gislaine Silva	Av. Nelson Dávila, 2345 - São José dos Campos	F	Mulher
7	Antônio Pereira	Rua Joaquim Nabuco, 18 - Jacareí	M	Homem
8	Jair Lopes	Rua Pedro XIII, Santa Isabel	M	Homem
.....				
43	Heidi Lima	Rua César Conceição, 12 - Santo Antônio do Pinhal	F	Mulher
44	Amílcar Júnior	Rua Senador Kennedy, 901 - São Paulo	M	Homem
45	Alexandro Duarte	Rua Maria Cíntia - Cruzeiro	M	Homem
46	Maurício dos Santos	Rua Marta Silva - Jacareí	M	Homem
47	Mary Ann Duarte	Av. Brasil, 6320 - São Paulo	F	Mulher
48	Gabriela Pereira	Rua Franco Silva, 1599 - São Paulo	F	Mulher
49	André César	Rua Militar, 349 - Rio de Janeiro	M	Homem
50	Edson Lopes	Av. Pedro Silva, 3047 - Rio de Janeiro	M	Homem

(50 row(s) affected)



■ Estrutura de repetição

- No T-SQL, o comando **WHILE** define uma condição para a execução repetida de uma instrução ou um bloco de instruções SQL. As instruções serão executadas repetidamente desde que a condição especificada seja verdadeira.
- A execução de instruções no loop **WHILE** pode ser controlada internamente ao loop com as palavras-chave **BREAK** e **CONTINUE**.
- O escopo dos comandos a serem realizados é definido utilizando as palavras-chaves **BEGIN** e **END**.



■ Exemplo WHILE

-- Exemplo de um contador básico utilizando WHILE

```
DECLARE @i AS INT
```

```
SET @i = 1
```

```
WHILE @i <= 10 ←
```

```
    BEGIN
```

```
        PRINT 'Valor de i: ' + CAST(@i AS CHAR)
```

```
        SET @i = @i + 1
```

```
    END
```

```
GO
```



■ Exemplo WHILE

Valor de i: 1

Valor de i: 2

Valor de i: 3

Valor de i: 4

Valor de i: 5

Valor de i: 6

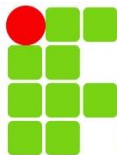
Valor de i: 7

Valor de i: 8

Valor de i: 9

Valor de i: 10





■ Exemplo WHILE

-- Criação de tabuada com WHILE

```
DECLARE @quantidade AS INT = 5,  
        @total      AS INT = 1,  
        @contador   AS INT,  
        @limite     AS INT = 10 ←
```

-- Loop que controla o total de tabuadas

```
WHILE @total <= @quantidade  
BEGIN
```

-- Imprime um cabeçalho

```
PRINT 'Tabuada do ' + CAST(@total AS  
VARCHAR(5))
```

```
PRINT REPLICATE('-', 13)
```



■ Exemplo WHILE

-- Define o valor inicial para cada tabuada

SET @contador = 0

-- Loop que controla a criação da tabuada

WHILE @contador <= @limite

BEGIN

-- Imprime a tabuada

PRINT CAST(@total AS VARCHAR(5))

+ ' X '

+ CAST(@contador AS VARCHAR(5))

+ ' = '

+ CAST((@total * @contador) AS
VARCHAR(5))



■ Exemplo WHILE

```
-- Atualiza o multiplicador
SET @contador += 1

END

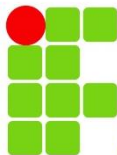
-- Atualiza a variável que controla a
-- quantidade de tabuadas
SET @total += 1

-- Imprime uma linha em branco
PRINT ' '

END

GO
```





■ Exemplo WHILE

```
-- Criação de tabuada com WHILE
DECLARE @quantidade AS INT = 5,
        @total      AS INT = 1,
        @contador   AS INT,
        @limite     AS INT = 10

-- Loop que controla o total de tabuadas
WHILE @total <= @quantidade
BEGIN
    -- Imprime um cabeçalho
    PRINT 'Tabuada do ' + CAST(@total AS VARCHAR(5))
    PRINT REPLICATE('-', 13)

    -- Define o valor inicial para cada tabuada
    SET @contador = 0

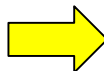
    -- Loop que controla a criação da tabuada
    WHILE @contador <= @limite
    BEGIN
        -- Imprime a tabuada
        PRINT CAST(@total AS VARCHAR(5))
          + ' X '
          + CAST(@contador AS VARCHAR(5))
          + ' = '
          + CAST((@total * @contador) AS VARCHAR(5))

        -- Atualiza o multiplicador
        SET @contador += 1
    END

    -- Atualiza a variável que controla a quantidade de tabuadas
    SET @total += 1

    -- Imprime uma linha em branco
    PRINT ''
END

GO
```



Tabuada do 1

1 X 0 = 0

1 X 1 = 1

1 X 2 = 2

1 X 3 = 3

Tabuada do 2

2 X 0 = 0

2 X 1 = 2

2 X 2 = 4

2 X 3 = 6



- **Na próxima aula veremos**
 - Stored Procedures e Functions.

