# **Exercícios de Fixação**

# **(7) Programação T-SQL**

# **(7.1) COMENTÁRIOS / NOCOUNT / ANSI\_NULLS / QUOTED\_IDENTIFIER**

**1)** No SQL Server, podemos utilizar os tipos de comentários abaixo:

-- 1) COMENTÁRIO PARA UMA LINHA - USAR O "--"

-- 2) COMENTÁRIO PARA UM BLOCO DE LINHAS - USAR O "/\* TEXTO \*/"

/\*

ESSE

É UM

COMENTÁRIO EM BLOCO

\*/

(V) Verdadeiro

(F) Falso

**2)** Ao habilitar o “SET NOCOUNT ON”, iremos deixar de exibir a quantidade de linhas que foram afetadas pela query.

SET NOCOUNT ON

GO

SELECT \* FROM Cliente

/\*

(6 rows affected)

\*/

(V) Verdadeiro

(F) Falso

**3)** Ao desabilitar o “SET ANSI\_NULLS OFF”, somente o script 2 irá retornar os registros com valor NULL.

SET ANSI\_NULLS OFF

GO

-- SCRIPT 1:

SELECT \*

FROM #TEMP\_ANSI\_NULLS

WHERE Nome = NULL

-- SCRIPT 2:

SELECT \*

FROM #TEMP\_ANSI\_NULLS

WHERE Nome IS NULL

(V) Verdadeiro

(F) Falso

**4)** Ao desabilitar o “SET QUOTED\_IDENTIFIER OFF”, também podemos representar uma string/texto utilizando as “aspas duplas”.

SET QUOTED\_IDENTIFIER OFF

SELECT 'texto' AS "Aspas\_Simples"

SELECT "texto" AS "Aspas\_Duplas"

(V) Verdadeiro

(F) Falso

# **(7.2) IDENTITY / SEQUENCE / COLUNAS CALCULADAS**

**1)** Marque a alternativa CORRETA:

(1) O IDENTITY garante que os registros na tabela serão únicos.

(2) Podemos definir o valor do “seed” e “increment” no IDENTITY.

(3) Ao fazer um INSERT, não precisamos informar o valor da coluna que possui o IDENTITY.

(4) Assim como o DELETE, o TRUNCATE TABLE não reseta o IDENTITY da tabela.

a) F, V, V, F

b) F, V, F, V

c) F, F, V, F

d) V, F, V, V

e) V, V, V, F

**2)** O comando abaixo será executado com sucesso?

CREATE TABLE #TEMP\_CLIENTE (

Id\_Cliente INT IDENTITY(1,1) NOT NULL,

Nm\_Cliente VARCHAR(100) NOT NULL,

Vl\_Salario NUMERIC (9,2) NOT NULL,

)

INSERT INTO #TEMP\_CLIENTE (Id\_Cliente, Nm\_Cliente, Vl\_Salario)

VALUES (1, 'Luiz Lima', 1000.00)

(V) Verdadeiro

(F) Falso

**3)** O comando abaixo será executado com sucesso?

CREATE TABLE #TEMP\_CLIENTE (

Id\_Cliente INT IDENTITY(1,1) NOT NULL,

Nm\_Cliente VARCHAR(100) NOT NULL,

Vl\_Salario NUMERIC (9,2) NOT NULL,

)

GO

SET IDENTITY\_INSERT #TEMP\_CLIENTE ON;

INSERT INTO #TEMP\_CLIENTE (Id\_Cliente, Nm\_Cliente, Vl\_Salario)

VALUES (1, 'Luiz Lima', 1000.00)

SET IDENTITY\_INSERT #TEMP\_CLIENTE OFF;

(V) Verdadeiro

(F) Falso

**4)** Marque a alternativa CORRETA:

(1) Podemos criar um SEQUENCE com o script abaixo:

CREATE SEQUENCE dbo.SeqID\_test

AS [INT]

START WITH 1

INCREMENT BY 1

(2) Podemos utilizar o comando abaixo para retornar o próximo valor do SEQUENCE.

SELECT NEXT VALUE FOR dbo.SeqID\_test

(3) Não podemos utilizar o mesmo SEQUENCE em tabelas diferentes.

(4) O script abaixo irá gerar um erro.

CREATE TABLE #TEMP\_CLIENTE (

Id\_Cliente INT DEFAULT NEXT VALUE FOR dbo.SeqID\_test,

Nm\_Cliente VARCHAR(100) NOT NULL,

Vl\_Salario NUMERIC (9,2) NOT NULL,

)

a) V, F, V, V

b) V, V, V, F

c) V, V, F, F

d) F, V, V, V

e) F, V, F, V

**5)** Explique para que serve a opção “CYCLE” no SEQUENCE abaixo:

CREATE SEQUENCE dbo.SeqID\_test

AS [INT]

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 5

CYCLE

SELECT NEXT VALUE FOR dbo.SeqID\_test

GO

**6)** Marque a alternativa CORRETA:

(1) Toda coluna calculada fica armazenada fisicamente junto com as outras colunas da tabela.

(2) Uma coluna calculada PERSISTED fica armazenada fisicamente. Com isso, temos um ganho de performance, pois ela não vai precisar ser calculada em tempo de execução.

(3) Podemos criar índices em uma coluna calculada PERSISTED.

(4) O comando abaixo adiciona a coluna calculada “Vl\_Total” na tabela “Estoque”.

ALTER TABLE Estoque

ADD Vl\_Total AS (Qtde\_Produto \* Vl\_Produto) PERSISTED

GO

a) F, V, V, V

b) F, V, F, V

c) F, F, V, V

d) V, F, V, F

e) V, V, F, V

**7)** Podemos criar uma coluna calculada “idade” como PERSISTED, pois dessa forma ela vai ser armazenada fisicamente.

ALTER TABLE [dbo].[Funcionario]

ADD Idade AS DATEDIFF(YEAR, Dt\_Nascimento, GETDATE()) PERSISTED

GO

(V) Verdadeiro

(F) Falso

**8)** Podemos utilizar uma função em uma coluna calculada.

ALTER TABLE Estoque

ADD Vl\_Total AS [dbo].[fncCalculaValorTotal](Qtde\_Produto, Vl\_Produto)

GO

(V) Verdadeiro

(F) Falso

# **(7.3) TRY ... CATCH / DYNAMIC SQL**

**1)** O segundo SELECT dentro do bloco TRY será executado normalmente.

BEGIN TRY

SELECT 15 / 0

SELECT 'OI! ESTOU DENTRO DO BLOCO "TRY"!'

END TRY

BEGIN CATCH

SELECT 'OI! ESTOU DENTRO DO BLOCO "CATCH"!'

END CATCH

GO

(V) Verdadeiro

(F) Falso

**2)** O comando “THROW” pode ser utilizado para gerar uma exceção e retornar o erro.

BEGIN TRY

SELECT 15 / 0

SELECT 'OI! ESTOU DENTRO DO BLOCO "TRY"!'

END TRY

BEGIN CATCH

THROW

END CATCH

GO

(V) Verdadeiro

(F) Falso

**3)** O SELECT dentro do bloco do CATCH será executado normalmente.

BEGIN TRY

SELECT 1

END TRY

BEGIN CATCH

SELECT 'OI! ESTOU DENTRO DO BLOCO "CATCH"!'

END CATCH

GO

(V) Verdadeiro

(F) Falso

**4)** No SQL Server temos algumas funções que retornam informações sobre os erros.

BEGIN TRY

SELECT 15 / 0

SELECT 'OI! ESTOU DENTRO DO BLOCO "TRY"!'

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS [Error\_Number],

ERROR\_MESSAGE() AS [Error\_Message],

ERROR\_SEVERITY() AS [Error\_Severity],

ERROR\_STATE() AS [Error\_State],

ERROR\_LINE() AS [Error\_Line],

ERROR\_PROCEDURE() AS [Error\_Procedure]

END CATCH

GO

(V) Verdadeiro

(F) Falso

**5)** Os dois comandos abaixo irão retornar o mesmo resultado.

EXEC sp\_executesql N'SELECT \* FROM Cliente'

EXEC (N'SELECT \* FROM Cliente')

(V) Verdadeiro

(F) Falso

**6)** É uma boa prática utilizar a procedure “sp\_executesql” ao invés do “EXEC (@querysql)”, pois com a “sp\_executesql” podemos especificar os parâmetros e evitar também ataques via SQL Injection.

EXEC sp\_executesql

@stmt = N'SELECT \* FROM Cliente WHERE Nm\_Cliente = @Nm\_Cliente',

@params = N'@Nm\_Cliente AS VARCHAR(100)',

@Nm\_Cliente = 'Fabrício Lima'

(V) Verdadeiro

(F) Falso

**7)** Explique o que o comando abaixo faz:

EXEC sp\_msforeachdb 'USE [?]; SELECT DB\_NAME()'

**8)** Cite um possível problema de segurança que podemos ter ao utilizar a procedure abaixo, e também uma possível solução:

CREATE PROCEDURE stpRetornaCliente

@Nm\_Cliente VARCHAR(100)

AS

BEGIN

DECLARE @Ds\_Comando NVARCHAR(500) = ''

SELECT @Ds\_Comando += 'SELECT \* FROM Cliente WHERE Nm\_Cliente = ''' + @Nm\_Cliente + ''''

EXEC (@Ds\_Comando)

END

GO