# **Exercícios de Fixação**

# **(4) Estrutura de uma Query**

# **(4.1) Executando uma Query – Passo a Passo:**

**1)** Qual a ordem correta do Processamento Lógico de uma Query?

SELECT Id\_Loja, YEAR(Dt\_Venda) AS Nr\_Ano, COUNT(\*) AS Qt\_Vendas, SUM(Vl\_Venda) AS Vl\_Total

FROM Vendas

WHERE Id\_Cliente = 1

GROUP BY Id\_Loja, YEAR(Dt\_Venda)

HAVING COUNT(\*) > 1

ORDER BY Id\_Loja, Nr\_Ano

a) SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY

b) FROM, WHERE, SELECT, GROUP BY, HAVING, ORDER BY

c) FROM, WHERE, GROUP BY, SELECT, HAVING, ORDER BY

d) FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY

e) FROM, SELECT, WHERE, GROUP BY, HAVING, ORDER BY

**2)** Marque a alternativa CORRETA:

(1) O **“\*”** (asterisco) pode ser utilizado para retornar todas as colunas de uma tabela no SELECT.

(2) A sintaxe do comando abaixo está INCORRETA:

SELECT [colunas]

FROM [ServerName].[DatabaseName].[SchemaName].[ObjectName]

(3) A sintaxe do comando abaixo está CORRETA:

SELECT \* FROM Cliente

SELECT \* FROM [Cliente]

SELECT \* FROM "Cliente"

(4) Utilizar funções em colunas na clausula WHERE é uma boa prática e não afeta a performance.

a) V, V, V, F

b) V, F, F, F

c) V, F, V, F

d) F, F, V, F

e) F, V, V, F

**3)** Marque a alternativa CORRETA:

(1) No SQL Server temos algumas funções de agregação, por exemplo: COUNT, SUM, AVG, MIN, MAX.

(2) Todas as funções de agregação (COUNT, SUM, AVG, MIN, MAX) sempre ignoram os valores NULL.

(3) A fórmula da função “Média” é a seguinte: AVG() = SUM() / COUNT()

(4) A sintaxe do comando abaixo está CORRETA:

SELECT

Id\_Loja,

YEAR(Dt\_Venda) AS Nr\_Ano,

SUM(Vl\_Venda) AS Vl\_Total

FROM Vendas

WHERE Id\_Cliente = 1 AND Id\_Loja = 1

GROUP BY YEAR(Dt\_Venda)

a) V, F, V, F

b) V, F, V, V

c) V, V, V, F

d) F, V, F, V

e) F, V, V, F

**4)** Monte um SELECT para retornar o valor total das vendas agrupado por ano e ordenado de forma decrescente pelo valor total.

OBS: Utilizar a tabela que foi criada no arquivo "DEMO 4.0 - Cria Tabela Vendas.sql".

Resultado esperado (nome das colunas): Ano, Valor Total

USE Treinamento\_TSQL

SELECT \*

FROM Vendas

**5)** Monte um SELECT para retornar o valor total das vendas agrupado por ano e ordenado de forma decrescente pelo valor total. Além disso, retorne apenas os anos que tiveram mais de R$3.000,00 em vendas.

OBS: Utilizar a tabela que foi criada no arquivo "DEMO 4.0 - Cria Tabela Vendas.sql".

Resultado esperado (nome das colunas): Ano, Valor Total

USE Treinamento\_TSQL

SELECT \*

FROM Vendas

**6)** Os dois scripts abaixo fazem a mesma coisa e sempre retornam o mesmo resultado?

-- SCRIPT 1:

SELECT YEAR(Dt\_Venda) AS Ano, SUM(Vl\_Venda) AS Vl\_Total

FROM Vendas

GROUP BY YEAR(Dt\_Venda)

HAVING SUM(Vl\_Venda) > 3000

ORDER BY Ano DESC

-- SCRIPT 2:

SELECT YEAR(Dt\_Venda) AS Ano, SUM(Vl\_Venda) AS Vl\_Total

FROM Vendas

WHERE Vl\_Venda > 3000

GROUP BY YEAR(Dt\_Venda)

ORDER BY Ano DESC

(V) Verdadeiro

(F) Falso

**7)** Marque a alternativa CORRETA:

(1) A sintaxe do comando abaixo está CORRETA.

SELECT \*

FROM Cliente

ORDER BY Vl\_Salario DESC

(2) A sintaxe do comando abaixo está INCORRETA, pois não é possível utilizar uma scalar function no ORDER BY, uma vez que ele é utilizado apenas para ordenar o resultado.

SELECT \*, [dbo].[fncCalculaAumentoSalario] (Vl\_Salario, 1.1) AS Vl\_Salario\_Novo

FROM Cliente

ORDER BY [dbo].[fncCalculaAumentoSalario] (Vl\_Salario, 1.1) DESC

(3) Na ordem de execução lógica de uma query, o SQL Server irá executar o ORDER BY antes do SELECT.

(4) É uma boa prática utilizar um número ordinal no ORDER BY, pois isso facilita a manutenção do script.

SELECT Id\_Cliente, Vl\_Salario

FROM Cliente

ORDER BY 2 DESC

(5) Os dados temporários das ordenações (ORDER BY) ficam armazenados na database TEMPDB e não na database que o script está sendo executado.

a) V, F, F, F, V

b) V, F, V, V, F

c) V, V, V, V, F

d) F, F, F, F, V

**8)** Marque a alternativa CORRETA sobre o comando abaixo:

SELECT Nm\_Cliente

FROM Cliente

ORDER BY Nm\_Cliente

OPTION (MAXDOP 1, RECOMPILE)

(1) A sintaxe do comando acima está INCORRETA, pois o OPTION deveria ser antes do ORDER BY.

(2) Não é possível utilizar esse tipo de comando (OPTION) no SQL Server.

a) V, V

b) V, F

c) F, F

d) F, V

**9)** Explique qual o ERRO do comando abaixo:

USE Treinamento\_TSQL

SELECT

[Id\_Cliente],

[Nm\_Cliente],

YEAR([Dt\_Nascimento]) AS [Ano]

FROM [dbo].[Cliente]

WHERE Ano >= 1970

**10)** Dado a tabela abaixo:

CREATE TABLE #Cliente (

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

Nm\_Cliente VARCHAR(100) NOT NULL,

Vl\_Salario NUMERIC(9,2) NULL

)

INSERT INTO #Cliente

VALUES ('Fabricio Lima', 50000.00),

('Luiz Lima', 5000.00),

('Dirceu Resende', NULL)

Qual cliente será retornado primeiro no script abaixo:

SELECT \*

FROM #Cliente

ORDER BY Vl\_Salario

a) Fabricio Lima

b) Luiz Lima

c) Dirceu Resende

d) Vai dar um erro, pois a sintaxe do comando está incorreta.

**11)** Monte um SELECT para retornar o valor total das vendas, a quantidade e a média do valor das vendas para cada cliente. Por fim, ordenar o resultado de forma decrescente pelo valor total.

OBS: Utilizar a tabela que foi criada no arquivo "DEMO 4.0 - Cria Tabela Vendas.sql".

Resultado esperado (nome das colunas): ID Cliente, Valor Total, Qtde, Media

USE Treinamento\_TSQL

SELECT \*

FROM Vendas

# **(4.2) JOINs – Unindo tabelas:**

**1)** Marque a alternativa CORRETA:

(1) O CROSS JOIN pode ser utilizado para fazer o Produto Cartesiano de cada elemento da Tabela A com os elementos da Tabela B.

SELECT \*

FROM #TABELA\_A AS A

CROSS JOIN #TABELA\_B AS B

ORDER BY A.ID, B.ID

(2) O INNER JOIN pode ser utilizado para retornar apenas os elementos que existem nas duas tabelas.

SELECT \*

FROM #TABELA\_A A

INNER JOIN #TABELA\_B B ON A.ID = B.ID

ORDER BY A.ID, B.ID

(3) Não podemos fazer um JOIN com a própria tabela. Portanto, o script abaixo está INCORRETO.

SELECT \*

FROM [dbo].[Funcionario] F

JOIN [dbo].[Funcionario] G ON F.Id\_Gerente = G.Id\_Funcionario

(4) Os valores NULL na coluna “ID” serão retornados normalmente no SELECT abaixo:

SELECT \*

FROM #TABELA\_A A

INNER JOIN #TABELA\_B B ON A.ID = B.ID

ORDER BY A.ID, B.ID

a) F, F, V, V

b) F, V, F, V

c) V, V, F, V

d) V, V, F, F

e) V, V, V, V

**2)** Marque a alternativa CORRETA:

(1) Os dois SELECTs abaixo fazem a mesma coisa:

SELECT \*

FROM #TABELA\_B B1

JOIN #TABELA\_B B2 ON (B1.ID = B2.ID OR (B1.ID IS NULL AND B2.ID IS NULL))

SELECT \*

FROM #TABELA\_B B1

JOIN #TABELA\_B B2 ON ISNULL(B1.ID,'') = ISNULL(B2.ID,'')

(2) O JOIN faz um DISTINCT implicitamente para eliminar os resultados duplicados do SELECT abaixo:

SELECT \*

FROM #TABELA\_A A

JOIN #TABELA\_DUPLICADOS B ON A.ID = B.ID

(3) O SELECT abaixo irá retornar apenas os IDs que existem na Tabela A e não existem na Tabela B.

SELECT \*

FROM #TABELA\_A A

LEFT JOIN #TABELA\_B B ON A.ID = B.ID

(4) Os dois SELECTs abaixo retornam o mesmo resultado.

SELECT A.\*

FROM #TABELA\_A A

LEFT JOIN #TABELA\_B B ON A.ID = B.ID

WHERE B.ID IS NULL

SELECT \* FROM #TABELA\_A

EXCEPT

SELECT \* FROM #TABELA\_B

a) V, V, F, V

b) V, F, F, V

c) V, F, V, V

d) F, F, V, V

e) F, V, F, V

**3)** Os dois SELECTs abaixo retornam o mesmo resultado:

SELECT \*

FROM #TABELA\_A A

LEFT JOIN #TABELA\_B B ON A.ID = B.ID

WHERE B.ID IS NULL

SELECT \*

FROM #TABELA\_B B

LEFT JOIN #TABELA\_A A ON A.ID = B.ID

WHERE A.ID IS NULL

(V) Verdadeiro

(F) Falso

**4)** Dado as duas tabelas abaixo:

CREATE TABLE #TABELA\_A (

ID INT NULL

)

INSERT INTO #TABELA\_A (ID) VALUES (1),(2),(3),(4)

CREATE TABLE #TABELA\_B (

ID INT NULL

)

INSERT INTO #TABELA\_B (ID) VALUES (3),(4),(5),(6),(NULL)

-- VALIDA OS DADOS DAS TABELAS

SELECT \* FROM #TABELA\_A

SELECT \* FROM #TABELA\_B

OBS: DEVE USAR “JOINS” OBRIGATORIAMENTE!

(1) Faça um SELECT que retorna apenas a coluna ID da Tabela A, sendo que eles também pertençam a Tabela B.

(2) Faça um SELECT que retorna apenas a coluna ID da Tabela A, sendo que eles NÃO pertençam a Tabela B.

(3) Faça um SELECT que retorna apenas a coluna ID da Tabela B, sendo que eles NÃO pertençam a Tabela A.

**5)** A sintaxe do comando abaixo está INCORRETA.

SELECT \*

FROM #TABELA\_A A, #TABELA\_B B

WHERE A.ID = B.ID

ORDER BY A.ID, B.ID

(V) Verdadeiro

(F) Falso

**6)** A sintaxe do comando abaixo está CORRETA.

SELECT \*

FROM #TABELA\_A A

JOIN (

SELECT \*

FROM #TABELA\_B

) B ON A.ID = B.ID

ORDER BY A.ID, B.ID

(V) Verdadeiro

(F) Falso

**7)** Explique a diferença entre os dois scripts abaixo:

-- SCRIPT 1:

SELECT \*

FROM #TABELA\_A A1

INNER JOIN #TABELA\_A A2 ON A1.ID = A2.ID

INNER JOIN #TABELA\_A A3 ON A1.ID = A3.ID

INNER JOIN #TABELA\_A A4 ON A1.ID = A4.ID

LEFT JOIN #TABELA\_B A5 ON A1.ID = A5.ID

ORDER BY A1.ID

-- SCRIPT 2:

SELECT \*

FROM #TABELA\_A A1

INNER JOIN #TABELA\_A A2 ON A1.ID = A2.ID

INNER JOIN #TABELA\_A A3 ON A1.ID = A3.ID

INNER JOIN #TABELA\_A A4 ON A1.ID = A4.ID

INNER JOIN #TABELA\_B A5 ON A1.ID = A5.ID

ORDER BY A1.ID

**8)** Ao utilizar um JOIN, é uma boa prática não especificar o alias da tabela nas colunas utilizadas no SELECT.

SELECT ID

FROM #TABELA\_A A1

INNER JOIN #TABELA\_A A2 ON A1.ID = A2.ID

INNER JOIN #TABELA\_A A3 ON A1.ID = A3.ID

INNER JOIN #TABELA\_A A4 ON A1.ID = A4.ID

LEFT JOIN #TABELA\_B A5 ON A1.ID = A5.ID

ORDER BY A1.ID

/\*

Msg 209, Level 16, State 1, Line 214

Ambiguous column name 'ID'.

\*/

a) Verdadeiro

b) Falso

# **(4.3) UNION / DISTINCT / TOP:**

**1)** Marque a alternativa CORRETA:

(1) O “DISTINCT” é executado na fase lógica do ORDER BY, já o “TOP” é executado na fase lógica do SELECT.

(2) O comando abaixo irá retornar 5% dos registros da tabela “Vendas”.

SELECT TOP 5 PERCENT \* FROM Vendas ORDER BY Vl\_Venda

(3) O “OFFSET-FETCH” é muito útil quando precisar fazer “paginação” no resultado de uma query.

(4) Podemos utilizar o “TOP WITH TIES” (empate) para retornar todas as outras linhas que tiverem o mesmo valor que o último registro do TOP. Segue abaixo um script de exemplo:

SELECT TOP 5 WITH TIES \*

FROM Vendas

ORDER BY Vl\_Venda

a) F, V, V, V

b) F, V, F, F

c) F, F, V, V

d) V, F, V, F

e) V, V, F, V

**2)** Explique o que o script abaixo faz e qual será o seu resultado:

-- INTRODUÇÃO: APENAS PARA VISUALIZAR OS DADOS DA TABELA "Vendas"

SELECT \* FROM Vendas ORDER BY Id\_Venda

SELECT \*

FROM Vendas

ORDER BY Id\_Venda

OFFSET 10 ROWS FETCH NEXT 5 ROWS ONLY

**3)** Explique o que o script abaixo faz e qual será o seu resultado:

USE Treinamento\_TSQL

-- INTRODUÇÃO: APENAS PARA VISUALIZAR OS DADOS DA TABELA "Vendas"

SELECT \* FROM Vendas ORDER BY Vl\_Venda

-- SCRIPT 1:

SELECT TOP 5 \*

FROM Vendas

ORDER BY Vl\_Venda

-- SCRIPT 2:

SELECT TOP 5 WITH TIES \*

FROM Vendas

ORDER BY Vl\_Venda

**4)** Marque a alternativa CORRETA:

SELECT \*

FROM Vendas

ORDER BY Id\_Venda

OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY

a) Irá desconsiderar as 10 primeiras linhas e retornar as próximas 20.

b) Irá desconsiderar as 20 primeiras linhas e retornar as próximas 10.

c) Irá dar um erro, pois a sintaxe está incorreta.

d) Não sei Luiz, mim explica esse ai de novo por favor.

**5)** Dado as duas tabelas abaixo, é CORRETO afirmar que:

CREATE TABLE #TabelaA (

ID INT NOT NULL

)

INSERT INTO #TabelaA (ID) VALUES (1),(2),(3),(4)

CREATE TABLE #TabelaB (

ID INT NOT NULL

)

INSERT INTO #TabelaB (ID) VALUES (3),(4),(5),(6)

(1) O Resultado será: 1, 2, 3, 4, 5, 6

SELECT \* FROM #TabelaA

UNION

SELECT \* FROM #TabelaB

(2) O Resultado será: 1, 2, 3, 4, 5, 6

SELECT \* FROM #TabelaA

UNION ALL

SELECT \* FROM #TabelaB

(3) O Resultado será: 3, 4

SELECT \* FROM #TabelaA

INTERSECT

SELECT \* FROM #TabelaB

(4) O Resultado será: 5, 6

SELECT \* FROM #TabelaA

EXCEPT

SELECT \* FROM #TabelaB

(5) O Resultado será: 1, 2

SELECT \* FROM #TabelaB

EXCEPT

SELECT \* FROM #TabelaA

a) V, F, V, F, F

b) V, F, V, V, F

c) V, F, F, F, F

d) F, V, V, V, V

e) F, V, V, F, F

**6)** Dado as duas tabelas abaixo, é CORRETO afirmar que:

CREATE TABLE #TabelaA (

ID INT NULL

)

INSERT INTO #TabelaA (ID) VALUES (1),(NULL)

CREATE TABLE #TabelaB (

ID INT NULL

)

INSERT INTO #TabelaB (ID) VALUES (1),(2),(NULL)

O resultado do comando abaixo será?

SELECT \* FROM #TabelaA

INTERSECT

SELECT \* FROM #TabelaB

ORDER BY ID

a) 1

b) 1, 2

c) 2

d) NULL, 1

e) 1, NULL

# **(4.4) SUBQUERY, EXISTS, CTE, APPLY, LIKE, PIVOT / LINKED SERVER:**

**1)** Marque a alternativa CORRETA:

(1) Subquery é uma consulta que está aninhada dentro de outra consulta.

(2) As duas queries abaixo retornam o mesmo resultado.

-- QUERY 1

SELECT

Id\_Cliente,

Nm\_Cliente,

Dt\_Nascimento,

-- SUBQUERY

(

SELECT SUM(Vl\_Venda)

FROM Vendas V

WHERE V.Id\_Cliente = C.Id\_Cliente

) AS Vl\_Total

FROM Cliente C

-- QUERY 2

SELECT

Id\_Cliente,

Nm\_Cliente,

Dt\_Nascimento,

-- SUBQUERY

(

SELECT SUM(Vl\_Venda)

FROM Vendas V

) AS Vl\_Total

FROM Cliente C

(3) O script abaixo está INCORRETO.

SELECT \*

FROM (

SELECT \*

FROM Cliente

) AS A

(4) Sabendo que a tabela “Vendas” possui vários registros, o script abaixo irá executar normalmente.

DECLARE @Vl\_Total NUMERIC(9,2)

SELECT @Vl\_Total = (

SELECT Vl\_Venda

FROM Vendas

)

SELECT @Vl\_Total AS Vl\_Total

a) V, F, F, F

b) V, V, V, F

c) V, F, V, V

d) F, F, F, V

**2)** Marque a alternativa CORRETA:

(1) No trecho de código abaixo, a mensagem será exibida somente se o SELECT dentro do IF EXISTS retornar pelo menos um registro.

IF EXISTS (SELECT TOP 1 Id\_Cliente FROM Cliente WHERE Nm\_Cliente = @Nm\_Cliente)

SELECT 'EXISTE UM USUÁRIO COM O NOME: "' + @Nm\_Cliente + '"!’

(2) O script abaixo está INCORRETO e vai gerar um ERRO:

IF EXISTS(SELECT 1)

BEGIN

SELECT 111111

END

(3) O script abaixo está INCORRETO e vai gerar um ERRO:

SELECT \*

FROM [dbo].[Cliente] C

WHERE EXISTS (SELECT TOP 1 Id\_Venda FROM Vendas V WHERE C.Id\_Cliente = V.Id\_Cliente)

(4) Os comandos abaixo sempre irão retornar os mesmos resultados:

SELECT \*

FROM [dbo].[Cliente] C

WHERE

NOT EXISTS (SELECT DISTINCT Id\_Cliente FROM #TEMP\_VENDAS V WHERE C.Id\_Cliente = V.Id\_Cliente)

SELECT \*

FROM [dbo].[Cliente] C

WHERE

Id\_Cliente NOT IN (SELECT DISTINCT Id\_Cliente FROM #TEMP\_VENDAS)

a) V, V, F, F

b) V, F, F, F

c) V, V, F, V

d) F, V, F, F

e) F, V, V, F

**3)** Marque a alternativa CORRETA:

(1) Podemos utilizar as CTEs (COMMON TABLE EXPRESSION) para implementar a Recursividade no SQL Server de forma ilimitada.

(2) Não podemos utilizar um UNION ALL em uma CTE, pois é permitido utilizar apenas um SELECT.

;WITH CTE\_RECURSIVA AS (

-- NIVEL ANCORA

SELECT F.\*

FROM Funcionario F

WHERE Id\_Gerente IS NULL

UNION ALL

-- DEMAIS NIVEIS

SELECT F.\*

FROM Funcionario F

JOIN CTE\_RECURSIVA CTE ON F.Id\_Gerente = CTE.Id\_Funcionario

)

SELECT \* FROM CTE\_RECURSIVA

(3) Podemos utilizar várias CTEs, conforme o exemplo abaixo:

;WITH CTE\_CLIENTE AS ( -- CTE 1

SELECT \* FROM Cliente

),

CTE\_VENDAS AS ( -- CTE 2

SELECT V.\*

FROM Vendas V

JOIN CTE\_CLIENTE C ON C.Id\_Cliente = V.Id\_Cliente

),

CTE\_VENDAS\_TOP10 AS ( -- CTE 3

SELECT TOP 10 \*

FROM CTE\_VENDAS

ORDER BY Vl\_Venda DESC

)

SELECT \*

FROM CTE\_VENDAS\_TOP10;

GO

a) V, V, F

b) V, F, V

c) F, F, V

d) F, V, F

**4)** Marque a alternativa CORRETA:

(1) O caractere “%” (percent) pode ser utilizado como “curinga” para um caractere no comando LIKE.

(2) Não podemos utilizar Expressões Regulares no comando LIKE.

(3) O SELECT abaixo retorna apenas os clientes cujo nome começa com “Luiz”.

SELECT Nm\_Cliente, Dt\_Nascimento

FROM [dbo].[Cliente]

WHERE Nm\_Cliente LIKE '%Luiz%'

(4) Os dois SELECTs abaixo sempre retornam os mesmos resultados, pois eles são equivalentes.

SELECT Nm\_Cliente

FROM Cliente

WHERE Nm\_Cliente LIKE '%\_%'

SELECT Nm\_Cliente

FROM Cliente

WHERE Nm\_Cliente LIKE '%[\_]%'

(5) O SELECT abaixo retorna os clientes que iniciam com a letra “A” até a letra “F”.

SELECT \*

FROM Cliente

WHERE Nm\_Cliente LIKE '[A-F]%'

ORDER BY Nm\_Cliente

a) V, F, F, F, V

b) V, F, V, F, V

c) F, F, V, V, V

d) F, V, V, F, V

e) F, F, F, F, V

**5)** Explique o que os scripts abaixo estão fazendo:

-- SCRIPT 1 - PIVOT

SELECT \*

INTO #TEMP\_PIVOT

FROM (

SELECT YEAR(Dt\_Venda) AS Ano, Vl\_Venda

FROM Vendas

) AS T

PIVOT (SUM(Vl\_Venda) FOR Ano IN ([2018], [2019], [2020], [2021])) AS Vl\_Total

SELECT \* FROM #TEMP\_PIVOT

-- SCRIPT 2 - UNPIVOT

SELECT Ano, Vl\_Total

FROM (

SELECT [2018], [2019], [2020], [2021] FROM #TEMP\_PIVOT

) AS T

UNPIVOT (Vl\_Total FOR Ano IN ([2018], [2019], [2020], [2021])) AS Vl\_Total

**6)** Marque a alternativa CORRETA:

(1) A sintaxe do comando abaixo está CORRETA:

SELECT \*

FROM [HPSPECTRE\SQL2017].[Traces].[dbo].[Cliente]

(2) Os dois comandos abaixo retornam o mesmo resultado:

SELECT \*

FROM [HPSPECTRE\SQL2017].[Traces].[dbo].[Cliente]

WHERE Nm\_Cliente = 'Luiz Lima'

SELECT \*

FROM OPENQUERY (

[HPSPECTRE\SQL2017],

'SELECT \* FROM .[Traces].[dbo].[Cliente] WHERE Nm\_Cliente = ''Luiz Lima'''

)

(3) Quando utilizamos um LINKED SERVER a query é processada no servidor remoto, enquanto o OPENQUERY processa a query localmente.

(4) O LINKED SERVER é mais recomendado para queries menores, enquanto o OPENQUERY é utilizado para queries maiores.

a) V, V, F, V

b) V, V, F, F

c) V, F, V, V

d) F, V, F, V

e) F, V, F, F

**7)** Marque a alternativa CORRETA:

(1) O operador CROSS APPLY é utilizado para fazer um “CROSS JOIN” entre duas tabelas, pois irá fazer a relação entre cada linha tabela à esquerda (Tabela A) com todas as linhas da tabela à direita (Tabela B).

(2) O operador OUTER APPLY retorna apenas as linhas que tiverem uma relação entre as duas tabelas.

(3) A sintaxe do comando abaixo está CORRETA:

SELECT \*

FROM [dbo].[Cliente] C

INNER JOIN [dbo].[fncRetornaVendasCliente](C.Id\_Cliente) V

(4) A sintaxe do comando abaixo está CORRETA:

SELECT \*

FROM [dbo].[Cliente] C

OUTER APPLY [dbo].[fncRetornaVendasCliente](C.Id\_Cliente) V

ORDER BY C.Id\_Cliente, V.Id\_Venda

(5) A sintaxe do comando abaixo está CORRETA:

SELECT \*

FROM [dbo].[Cliente] C

CROSS APPLY (

SELECT \*

FROM Vendas V

WHERE V.Id\_Cliente = C.Id\_Cliente

) V

ORDER BY C.Id\_Cliente, V.Id\_Venda

a) V, V, F, V, F

b) V, F, V, V, F

c) V, F, F, V, V

d) F, F, V, F, V

e) F, F, F, V, V