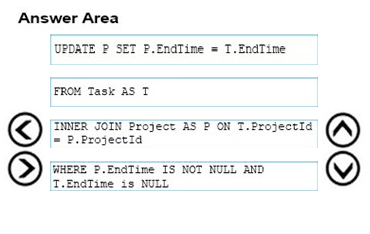
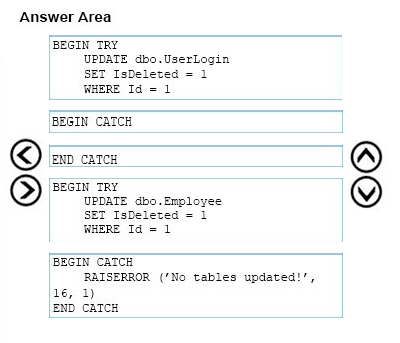
Querying Data with Transact-SQL v1.0 (70-761) – Topic 1

<https://www.itexams.com/exam/70-761>

1. Yes – **Errada –** Com o argumento XACT\_ABORT ON, a transação já é desconfirmada caso dê algum erro, tornando o código IF (XACT\_STATE()) <> 0 ROLLBACK incorreto; a transação já seria revertida de qualquer forma caso o erro acontecesse
2. No – **Certa –** Nesse caso estaria errado pois a declaração não retornaria o número do erro, apenas o armazenaria em uma variável
3. Yes – **Certa** – Caso algum erro acontecesse, nenhuma transação seria consolidada no banco e o erro 51000 seria mostrado em tela como solicitado
4. No – **Certa –** Aqui não daria certo pois falta um valor no INSERT; o numero de valores do INSERT deve ser o mesmo numero de colunas
5. Yes – **Errada –** Justificativa: As there are two separate INSERT INTO statements we cannot ensure that both or neither records are inserted.
6. Yes – **Certa** – Com o INSERT...VALUES nós asseguramos que o INSERT ocorra sem problemas
7. No – **Certa –** O tipo de retorno da função deveria ser nvarchar e não table
8. Yes – **Errada –** Neste caso, como o resultado da função será usado em uma VIEW, deveríamos ter usado a cláusula SCHEMABINDING
9. No – **Certa –** Não utilizar o TOP 1 no SELECT resulta em vários resultados no retorno
10. B – **Certa** – B era a única alternativa em que a contagem de ProjectId olhava para valores NULL, como pedia o enunciado
11. COALESCE() / LEFT JOIN – **Errada** – A alternativa certa deveria utilizar o **RIGHT JOIN** pois a tabela da direita (Projects) poderia ter valores NULL, mas a da esquerda (Tasks) não
12. **Errada**
13. **Errada –** A query correta deveria ser:



1. B – **Certa –** O segredo aqui estava no enunciado, criar algo que retorne um valor, pode ser usado em um SELECT, faz cálculos complexos, aceita parâmetros etc.
2. **Certa** – Nenhuma transação é executada. A primeira condição impede a primeira transação de executar. O bloco ELSE impede que a segunda transição seja executada
3. **Errada –** A opção que deveríamos utilizar é RAISERROR **WITH LOG**, e não NOWAIT
4. **Certa - Create procedure** 🡪 IF EXISTS 🡪 UPDATE SET companyname = @newname 🡪 IF NOT EXIST set custid = @custid 🡪 THROW ERROR
5. C – **Errada ­­**– As alternativas corretas são B e D. A = uma função que retorna uma tabela não pode ser chamada de dentro de uma VIEW com índice; C = uma SP não pode ser executada de dentro da VIEW
6. **Certa** – Begin TRY Update UserLogin 🡪 Begin CATCH 🡪 Begin TRY Update Employee 🡪 BEGIN CATCH RAISERROR. Minha sequencia ficou igual a do site:



1. **Errada** – A minha query:

select

AVG(NormalizedReading)

,dbo.NearestMountain(

from GroundSensors

where Tremor <> 0 and NormalizedReaging IS NOT NULL

A query certa:

SELECT

AVG(NormalizedReading)

,NearestMountain(**SensorID**)

FROM GroundSensors

WHERE TREMOR <> 0 AND NormalizedReading IS NOT NULL

**GROUP BY** NearestMountain(SensorID)

Para acertar faltou passar a **COLUNA SENSORID** como parâmetro da função. Não sabia que poderia passar uma coluna como parâmetro; e o **GROUP BY**

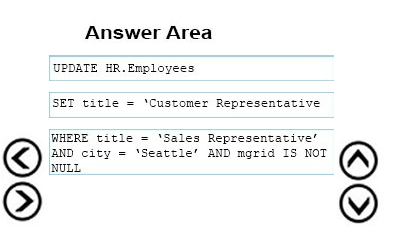
1. **Certa –** Minha query:

update HR.employees

set title = Customer Representative

where title = Sales Representative and city = Seattle and mgrid IS NOT NULL

Query correta:



1. TranslateName = Table-Valued, NearbyCities = Table-Valued. **Errada** – O retorno da primeira função é do tipo SCALAR (único), e não TABLE-VALUED
2. **Certa –** Minha query:

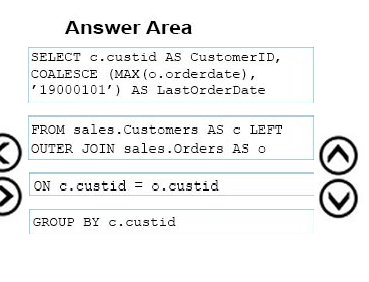
1. select c.custid As CustomerID, coalesce(MAX(o.orderdate),'19000101') as LastOrderDate

2. from sales.Customer AS c LEFT OUTER JOIN sales.Orders AS o

3. on c.custid = o.custid

4. group by c.custid

Query correta:



1. **Certa - custid** – coluna é identity; **postalcode** – coluna é NOT NULL, ou seja, deve ter um valor; **fax**/**region** – ambas colunas aceitam NULL, então a entrada de dados é opcional para ambas
2. **Certa** – Minha query:

delete from Sales.Orders

where CONVERT(varchar,OrderDate,112) < 20120101 AND ShippedDate IS NOT NULL

Query correta:

DELETE FROM Sales.Orders –

WHERE OrderDate < '2012-01-01' AND ShippedDate NOT NULL

1. **Certa –** Minha query:

SELECT TOP 3

P.FirstName + ' ' + P.LastName AS FullName

,S.SalesYTD

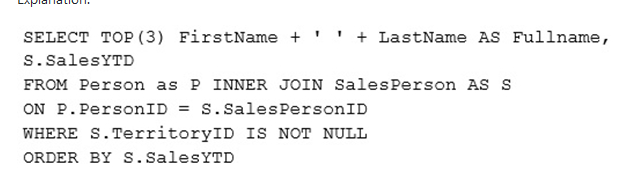
FROM Person AS P INNER JOIN SalesPerson AS S

ON P.PersonID = S.SalesPersonID

WHERE MIN(S.SalesYTD) AND TerritoryID IS NOT NULL

Faltou o ORDER BY na minha, mas no enunciado não pedia.

Query correta:



1. **Certa –** Minha query:

SELECT

Complaints.ComplaintID

,Persons.Name

FROM Contacts

LEFT JOIN Persons ON Persons.PersonID = Contacts.PersonID

LEFT JOIN Complaints ON Complaints.ComplaintID = Contacts.ComplaintID

Query correta:

SELECT

Complaints.ComplaintID

,Persons.Name

FROM Complaints

LEFT OUTER JOIN Contacts ON Complaints.ComplaintID = Contacts.ComplaintID

LEFT OUTER JOIN Persons ON Contacts.PersonID = Persons.PersonID

1. A – **Certa**

Minha query:

SELECT

C.CustomerID, ISNULL(SOH.SalesOrderID,0) AS OrderID

,ISNULL(MAX(OrderDate),'1900-01-01')

FROM Sales.Customer C

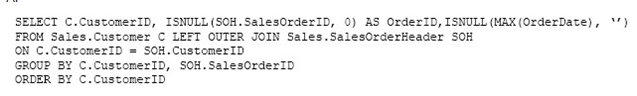
LEFT OUTER JOIN Sales.SalesOrderHeader SOH

ON C.CustomerID = SOH.CustomerID

GROUP BY C.CustomerID, SOH.SalesOrderID

ORDER BY C.CustomerID

Query correta:



1. D – **Certa**

Minha query:

UPDATE Customer

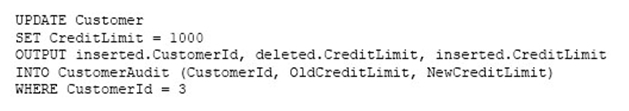
SET CreditLimit = 1000

OUTPUT CustomerID, deleted.CreditLimit, inserted.CreditLimit

INTO CustomerAudit (CustomerID, OldCreditLimit, NewCreditLimit)

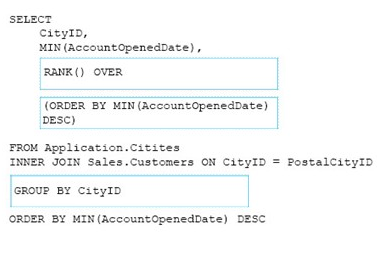
WHERE CustomerID = 3

Query correta:



1. **Certa –** RANK () OVER 🡪 (ORDER BY MIN(AccountOpenedDate) DESC) 🡪 GROUP BY CityID

Query correta:



1. B – **Certa**. Alternativa A ou B estariam certas.

**Explicação para A:** Um CROSS JOIN sem uma clausula WHERE funciona como plano cartesiano, neste caso, como existe uma clausula WHERE, o CROSS JOIN atua como um JOIN simples. **Explicação para B:** coloca uma condicional

INNER JOIN Application.Cities AS A ON A.CityID = IIF(IsOnCreditHold = 0, DeliveryCityID, PostalCityID)

1. A – **Errada.** Para restaurar um banco em uma data específica a cláusula **FOR SYSTEM\_TIME AS OF** utilizado como argumento do FROM é o ideal
2. **Errada**. Minha query:

SELECT

COUNT(Cust.CustomerId)

,CustCat.CustomerCategoryName

--1º ,CAST(Cust.AccountOpenedDate AS date)

FROM Sales.Customers AS Cust

INNER JOIN Sales.CustomerCategories AS CustCat

ON Cust.CustomerCategoryID = CustCat.CustomerCategoryID

--2º WHERE

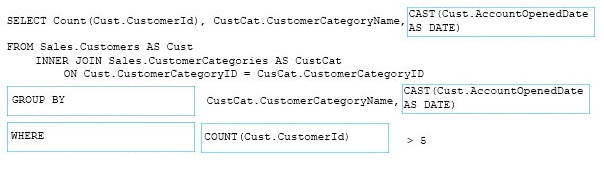
CustCat.CustomerCategoryName,

--3º MAX(Cust.CustomerId)

--4º HAVING

--5º COUNT(Cust.CustomerId) > 5

Query correta:



Essa tá estranha pq a resposta oficial tem um WHERE depois de um GROUP BY. Não é possível usar WHERE antes do GROUP BY. Se quiser filtrar dados depois do GROUP BY teria que usar o HAVING

1. **Errada**. Trabalhando com temporárias:

SELECT colunasPivoteadas

FROM (

SELECT

ColunaX,

ColunaY

FROM tabelaOrigem

) AS SourceTable

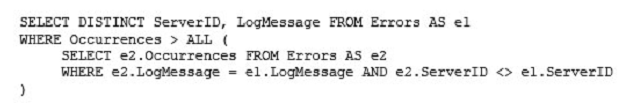
PIVOT (

SUM/AVG(ColunaSourceTable)

FOR ColunaSourceComLinhasASeremTransformadasEmColunas IN (Valores da Coluna FOR)

) AS CreditLimitTable

1. **Certa –** IN ( 🡪 WHERE 🡪 AND [IsOnCreditHold] = 0 🡪 )
2. A – **Certa –** A função COALESCE() oferece suporte a caracteres ANSI, enquanto a função ISNULL segue um padrão especifico do SQL Server
3. E – **Errada –** A resposta certa aqui seria VIEW e não Stored Procedure.
4. C – **Certa –** Uma função que retorna uma tabela seria o ideal para resolver esse problema
5. H – **Certa –** O método TRY\_CONVERT resolve essa situação.
6. **Certa –** CTE / Global temporary table / Local temporary table
7. C – **Errada.** Alternativa A é a correta:



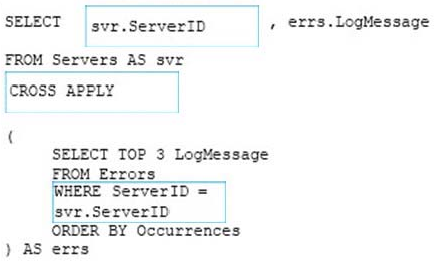
1. **Certa –** Meus segmentos:

1º svr.ServerID

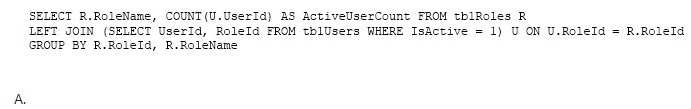
2º cross apply

3º WHERE serverId = svr.ServerID

Disposição correta do código:



1. A **– Certa.** Segmento correto:



As opções que tinham COUNT(\*) eventualmente teriam resultados na contagem, ignorando roles sem usuários ativos, não atendendo ao enunciado do exercício.

1. **Errada –**

Query correta:

SELECT CityID, QuestionID, RawCount

FROM (SELECT QuestionId, Tokyo, Boston, London, NewYork FROM RawSurvey) AS t1

UNPIVOT (RawCount FOR CityName IN (Tokyo, Boston, London, NewYork)) AS t2

JOIN Cities ON t2.CityName = t1.CityName

UNPIVOT Deve ser usado para rodar as colunas da tabela RawSurvey

1. **Errada –**

1 - bloco 2 - ok

2 - bloco 6 - x

3 - bloco 3 - x

4 - bloco 1 – ok

Eu inverti os blocos 6 e 3. O 3 deveria vir antes pois no 6 ele utiliza a CTE criada no bloco 3. Na prática eu utilizei uma query que ainda não havia sido criada.

ORDER BY somente na última etapa da CTE.