70-761$^{Q\&As}$

Querying Data with Transact-SQL

Pass Microsoft 70-761 Exam with 100% Guarantee

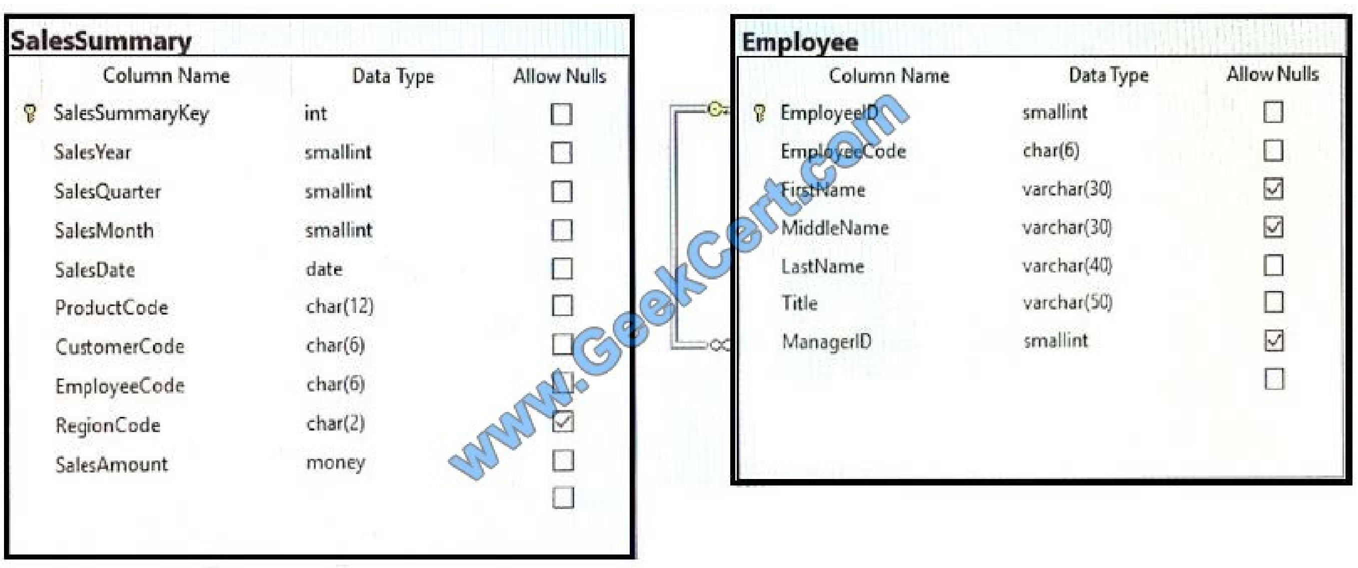
QUESTION 1

DRAG DROP

Note: This question is part of a series of questions that use the same scenario. For your convenience, the scenario is repeated in each question. Each question presents a different goal and answer choices, but the text of the scenario is exactly the same in each question in this series.

Start of repeated scenario

You have a database that contains the tables shown in the exhibit. (Click the Exhibit button.)



SalesSummary

Column Name	Data Type	Allow Nulls
SalesSummaryKey	int	☐
SalesYear	smallint	☐
SalesQuarter	smallint	☐
SalesMonth	smallint	☐
SalesDate	date	☐
ProductCode	char(12)	☐
CustomerCode	char(6)	☐
EmployeeCode	char(6)	☐
RegionCode	char(2)	✓
SalesAmount	money	☐
		☐

Employee

Column Name	Data Type	Allow Nulls
EmployeeID	smallint	☐
EmployeeCode	char(6)	☐
FirstName	varchar(30)	✓
MiddleName	varchar(30)	✓
LastName	varchar(40)	☐
Title	varchar(50)	☐
ManagerID	smallint	✓
		☐

You review the Employee table and make the following observations:

Every record has a value in the ManagerID except for the Chief Executive Officer (CEO).

The FirstName and MiddleName columns contain null values for some records.

The valid values for the Title column are Sales Representative manager, and CEO.

You review the SalesSummary table and make the following observations:

The ProductCode column contains two parts: The first five digits represent a product code, and the last seven digits represent the unit price. The unit price uses the following pattern: ####.##.

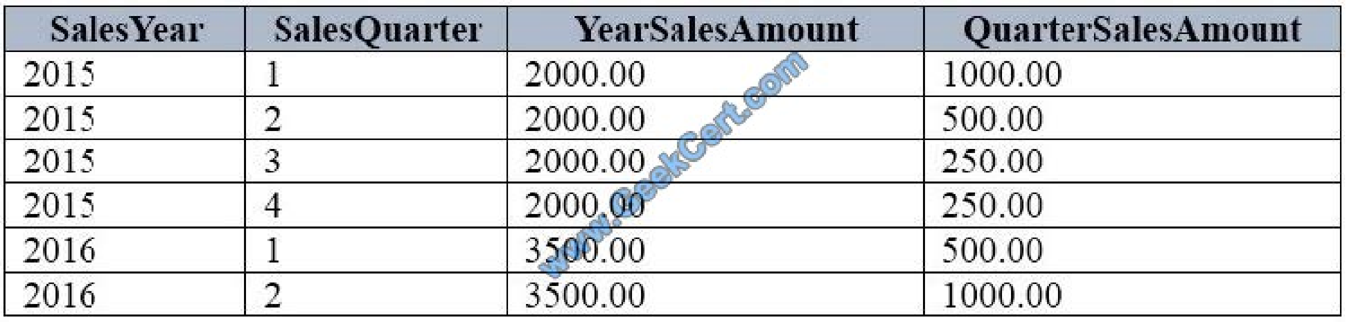
You observe that for many records, the unit price portion of the ProductCode column contains values.

The RegionCode column contains NULL for some records.

Sales data is only recorded for sales representatives.

You are developing a series of reports and procedures to support the business. Details for each report or procedure follow.

Sales Summary report: This report aggregates data by year and quarter. The report must resemble the following table.



SalesYear	SalesQuarter	YearSalesAmount	QuarterSalesAmount
2015	1	2000.00	1000.00
2015	2	2000.00	500.00
2015	3	2000.00	250.00
2015	4	2000.00	250.00
2016	1	3500.00	500.00
2016	2	3500.00	1000.00

Sales Manager report: This report lists each sales manager and the total sales amount for all employees that report to the sales manager.

Sales by Region report: This report lists the total sales amount by employee and by region. The report must include the following columns: EmployeeCode, MiddleName, LastName, RegionCode, and SalesAmount. If MiddleName is NULL, FirstName must be displayed. If both FirstName and MiddleName have null values, the world Unknown must be displayed/ If RegionCode is NULL, the word Unknown must be displayed.

Report1: This report joins data from SalesSummary with the Employee table and other tables. You plan to create an object to support Report1. The object has the following requirements:

be joinable with the SELECT statement that supplies data for the report

can be used multiple times with the SELECT statement for the report

be usable only with the SELECT statement for the report

not be saved as a permanent object

Report2: This report joins data from SalesSummary with the Employee table and other tables. You plan to create an object to support Report1. The object has the following requirements:

be joinable with the SELECT statement that supplies data for the report

can be used multiple times for this report and other reports

accept parameters

be saved as a permanent object

Sales Hierarchy report: This report aggregates rows, creates subtotal rows, and super-aggregates rows over the SalesAmount column in a single result-set. The report uses SaleYear, SaleQuarter, and SaleMonth as a hierarchy. The result set must not contain a grand total or cross-tabulation aggregate rows.

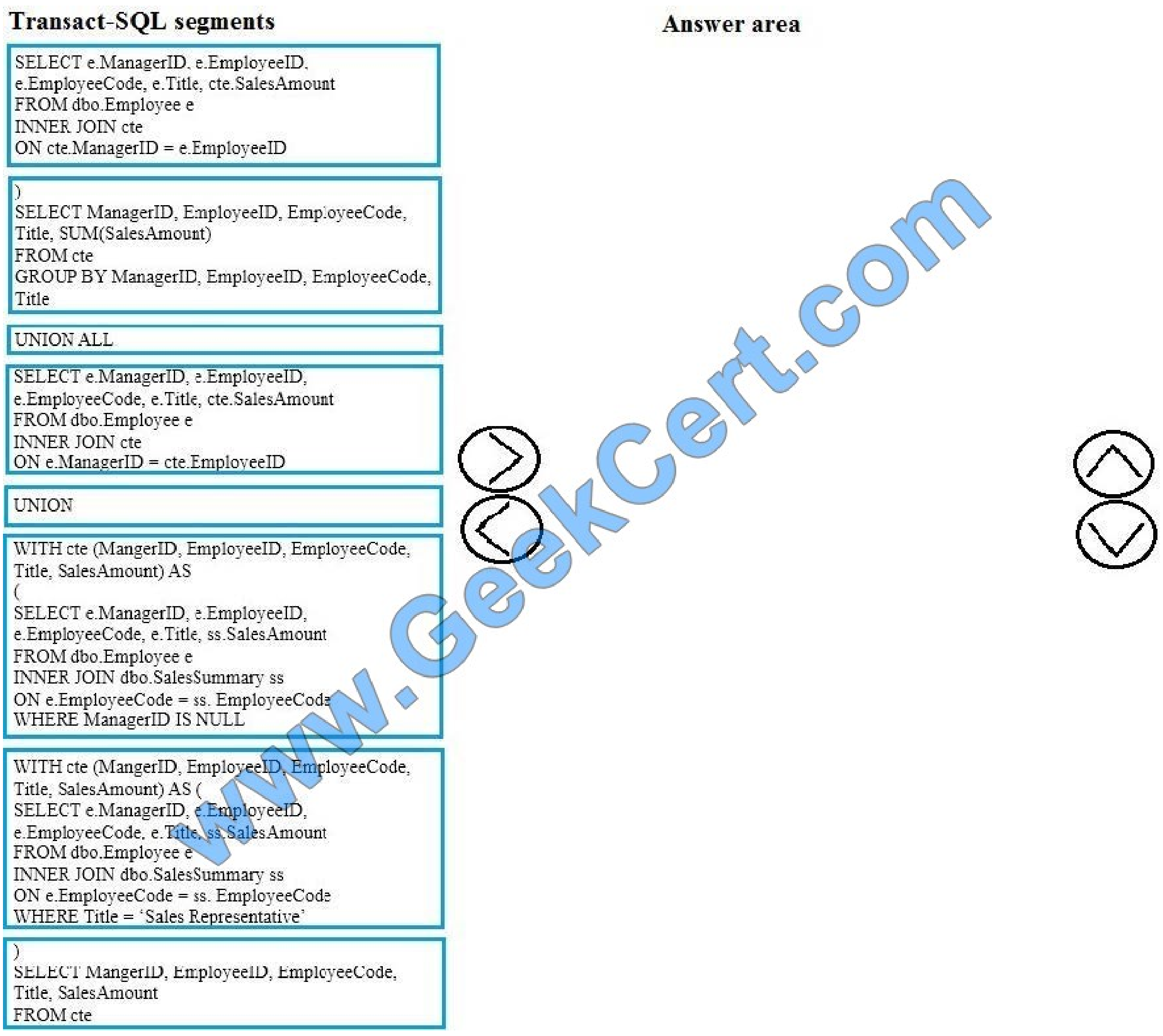
Current Price Stored Procedure: This stored procedure must return the unit price for a product when a product code is supplied. The unit price must include a dollar sign at the beginning. In addition, the unit price must contain a comma every three digits to the left of the decimal point, and must display two digits to the left of the decimal point. The stored procedure must not throw errors, even if the product code contains invalid data.

End of Repeated Scenario

You need to create the query for the Sales Managers report.

Which four Transact-SQL segments should you use to develop the solution? To answer, move the appropriate Transact-SQL segments from the list of Transact-SQL segments to the answer area and arrange them in the correct order.

Select and Place:



Transact-SQL segments

```
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, cte.SalesAmount
FROM dbo.Employee e
INNER JOIN cte
ON cte.ManagerID = e.EmployeeID
```

```
)
SELECT ManagerID, EmployeeID, EmployeeCode,
Title, SUM(SalesAmount)
FROM cte
GROUP BY ManagerID, EmployeeID, EmployeeCode,
Title
```

```
UNION ALL
```

```
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, cte.SalesAmount
FROM dbo.Employee e
INNER JOIN cte
ON e.ManagerID = cte.EmployeeID
```

```
UNION
```

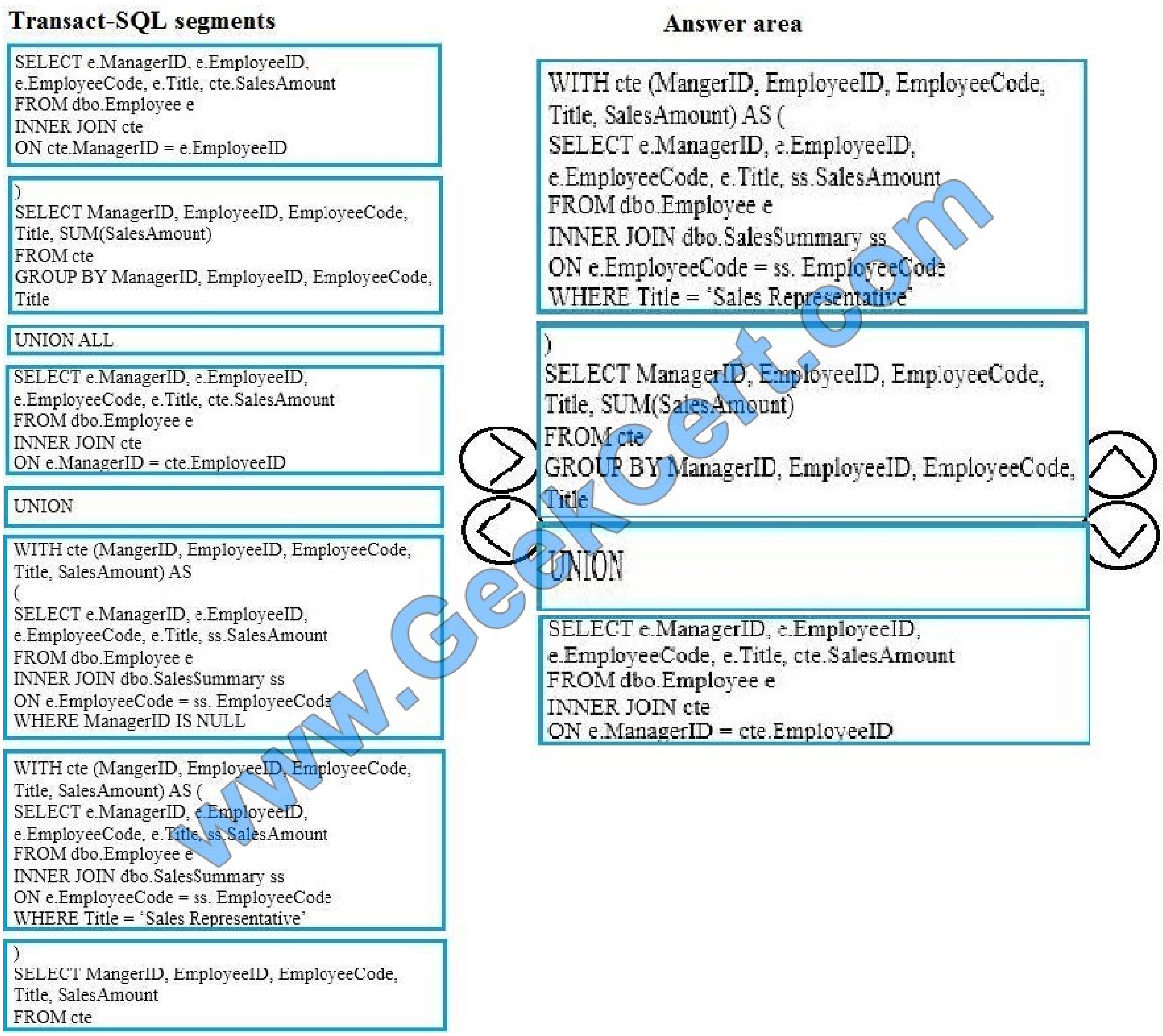
```
WITH cte (MangerID, EmployeeID, EmployeeCode,
Title, SalesAmount) AS
(
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, ss.SalesAmount
FROM dbo.Employee e
INNER JOIN dbo.SalesSummary ss
ON e.EmployeeCode = ss. EmployeeCode
WHERE ManagerID IS NULL
```

```
WITH cte (MangerID, EmployeeID, EmployeeCode,
Title, SalesAmount) AS (
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, ss.SalesAmount
FROM dbo.Employee e
INNER JOIN dbo.SalesSummary ss
ON e.EmployeeCode = ss. EmployeeCode
WHERE Title = 'Sales Representative'
```

```
)
SELECT MangerID, EmployeeID, EmployeeCode,
Title, SalesAmount
FROM cte
```

Answer area

Correct Answer:



Transact-SQL segments

```
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, cte.SalesAmount
FROM dbo.Employee e
INNER JOIN cte
ON cte.ManagerID = e.EmployeeID
```

```
)
SELECT ManagerID, EmployeeID, EmployeeCode,
Title, SUM(SalesAmount)
FROM cte
GROUP BY ManagerID, EmployeeID, EmployeeCode,
Title
```

```
UNION ALL
```

```
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, cte.SalesAmount
FROM dbo.Employee e
INNER JOIN cte
ON e.ManagerID = cte.EmployeeID
```

```
UNION
```

```
WITH cte (MangerID, EmployeeID, EmployeeCode,
Title, SalesAmount) AS
(
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, ss.SalesAmount
FROM dbo.Employee e
INNER JOIN dbo.SalesSummary ss
ON e.EmployeeCode = ss. EmployeeCode
WHERE ManagerID IS NULL
```

```
WITH cte (MangerID, EmployeeID, EmployeeCode,
Title, SalesAmount) AS (
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, ss.SalesAmount
FROM dbo.Employee e
INNER JOIN dbo.SalesSummary ss
ON e.EmployeeCode = ss. EmployeeCode
WHERE Title = 'Sales Representative'
```

```
)
SELECT MangerID, EmployeeID, EmployeeCode,
Title, SalesAmount
FROM cte
```

Answer area

```
WITH cte (MangerID, EmployeeID, EmployeeCode,
Title, SalesAmount) AS (
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, ss.SalesAmount
FROM dbo.Employee e
INNER JOIN dbo.SalesSummary ss
ON e.EmployeeCode = ss. EmployeeCode
WHERE Title = 'Sales Representative'
```

```
)
SELECT ManagerID, EmployeeID, EmployeeCode,
Title, SUM(SalesAmount)
FROM cte
GROUP BY ManagerID, EmployeeID, EmployeeCode,
Title
```

```
UNION
```

```
SELECT e.ManagerID, e.EmployeeID,
e.EmployeeCode, e.Title, cte.SalesAmount
FROM dbo.Employee e
INNER JOIN cte
ON e.ManagerID = cte.EmployeeID
```

From scenario: Sales Manager report: This report lists each sales manager and the total sales amount for all employees that report to the sales manager. Box 1:..WHERE Title=\\'Sales representative\\'

The valid values for the Title column are Sales Representative manager, and CEO.

First we define the CTE expression.

Note: A common table expression (CTE) can be thought of as a temporary result set that is defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement. A CTE is similar to a derived

table in that it is not stored as an object and lasts only for the duration of the query. Unlike a derived table, a CTE can be self-referencing and can be referenced multiple times in the same query.

Box 2:

Use the CTE expression one time.

Box 3: UNION

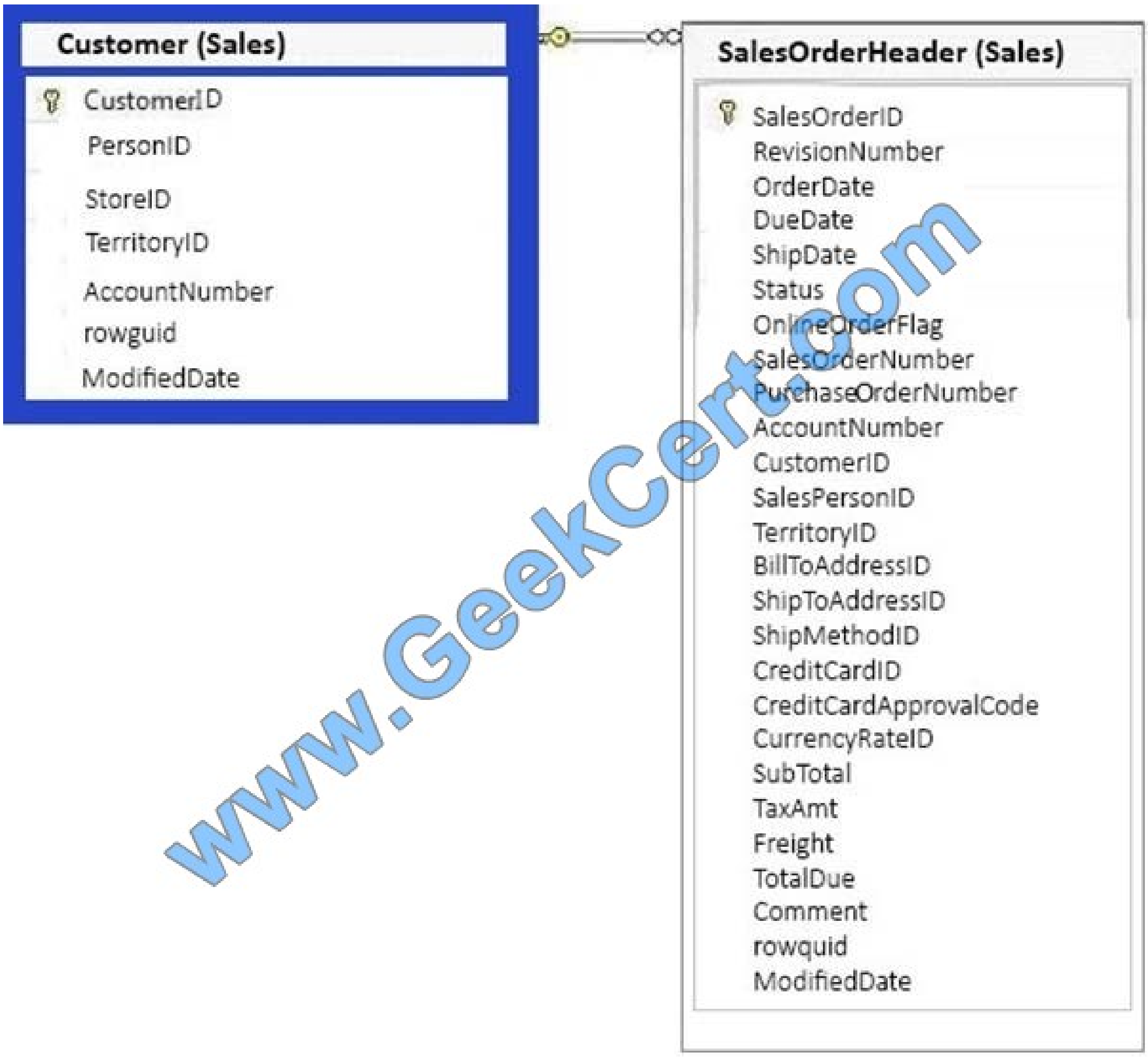
Box 4:

Use the CTE expression a second time.

References: https://technet.microsoft.com/en-us/library/ms190766(v=sql.105).aspx

QUESTION 2

You have a database that includes the tables shown in the exhibit. (Click the exhibit button.)



You need to create a list of all customers and the date that the customer placed their last order. For customers who have not placed orders, you must substitute a zero for the order ID and 01/01/1990 for the date. Which Transact-SQL statement should you run?

```
SELECT C.CustomerID, COALESCE(MAX(OrderDate), '19000101')
FROM Sales.Customer C LEFT OUTER JOIN Sales.SalesOrderHeader SOH
ON C.CustomerID = SOH.CustomerID
GROUP BY C.CustomerID
ORDER BY C.CustomerID
```

A.
```
SELECT C.CustomerID, COALESCE(MAX(OrderDate), '19000101')
FROM Sales.Customer C LEFT OUTER JOIN Sales.SalesOrderHeader SOH
ON C.CustomerID = SOH.CustomerID
GROUP BY C.CustomerID
ORDER BY C.CustomerID
```

B.
```
SELECT C.CustomerID, MAX(OrderDate)
FROM Sales.Customer C LEFT OUTER JOIN Sales.SalesOrderHeader SOH
ON C.CustomerID = SOH.CustomerID
GROUP BY C.CustomerID
ORDER BY C.CustomerID
```

C.
```
SELECT C.CustomerID, MAX(OrderDate)
FROM Sales.Customer C CROSS JOIN Sales.SalesOrderHeader SOH
GROUP BY C.CustomerID
ORDER BY C.CustomerID
```

D.
```
SELECT C.CustomerID, SOH.SalesOrderID, MAX(OrderDate)
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
```

A. B. C. D.

Correct Answer: A

COALESCE evaluates the arguments in order and returns the current value of the first expression that initially does not evaluate to NULL. References: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/coalesce-transact-sql

QUESTION 3

You have a table named Table1 that contains 200 million rows. Table1 contains a column named SaleDate that has a data type of DateTime2(3). Users report that the following query runs slowly.

```
Select SalesPerson, count(*)
FROM table1
Where year(SaleDate) = 2017
GROUP BY SalesPerson
```

You need to reduce the amount of time it takes to run the query.

What should you use to replace the WHERE statement?

A. WHERE SaleDate >= \\'2017-01-01\\' AND SaleDate

B. WHERE cast(SaleDate as varchar(10)) BETWEEN \\'2017-01-01\\' AND \\'2017-12-31\\'

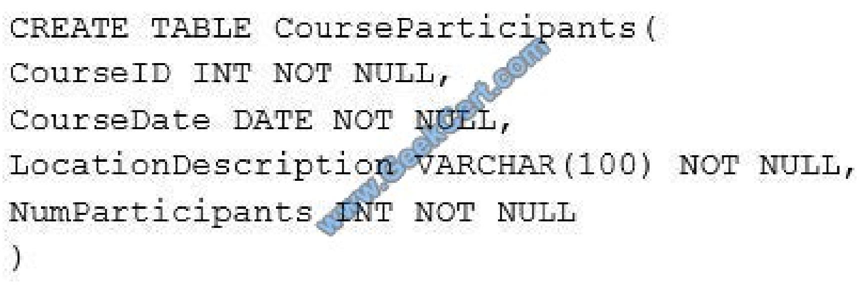
C. WHERE cast(SaleDate as date) BETWEEN \\'2017-01-01\\' AND \\'2017-12-31\\'

D. WHERE 2017 = year(SaleDate)

Correct Answer: C

References: https://docs.microsoft.com/en-us/sql/t-sql/queries/select-transact-sql?view=sql-server-2017

QUESTION 4



```
CREATE TABLE CourseParticipants(
CourseID INT NOT NULL,
CourseDate DATE NOT NULL,
LocationDescription VARCHAR(100) NOT NULL,
NumParticipants INT NOT NULL
)
```

You are a developing a solution to manage employee training records. You have the following Transact-SQL statement:

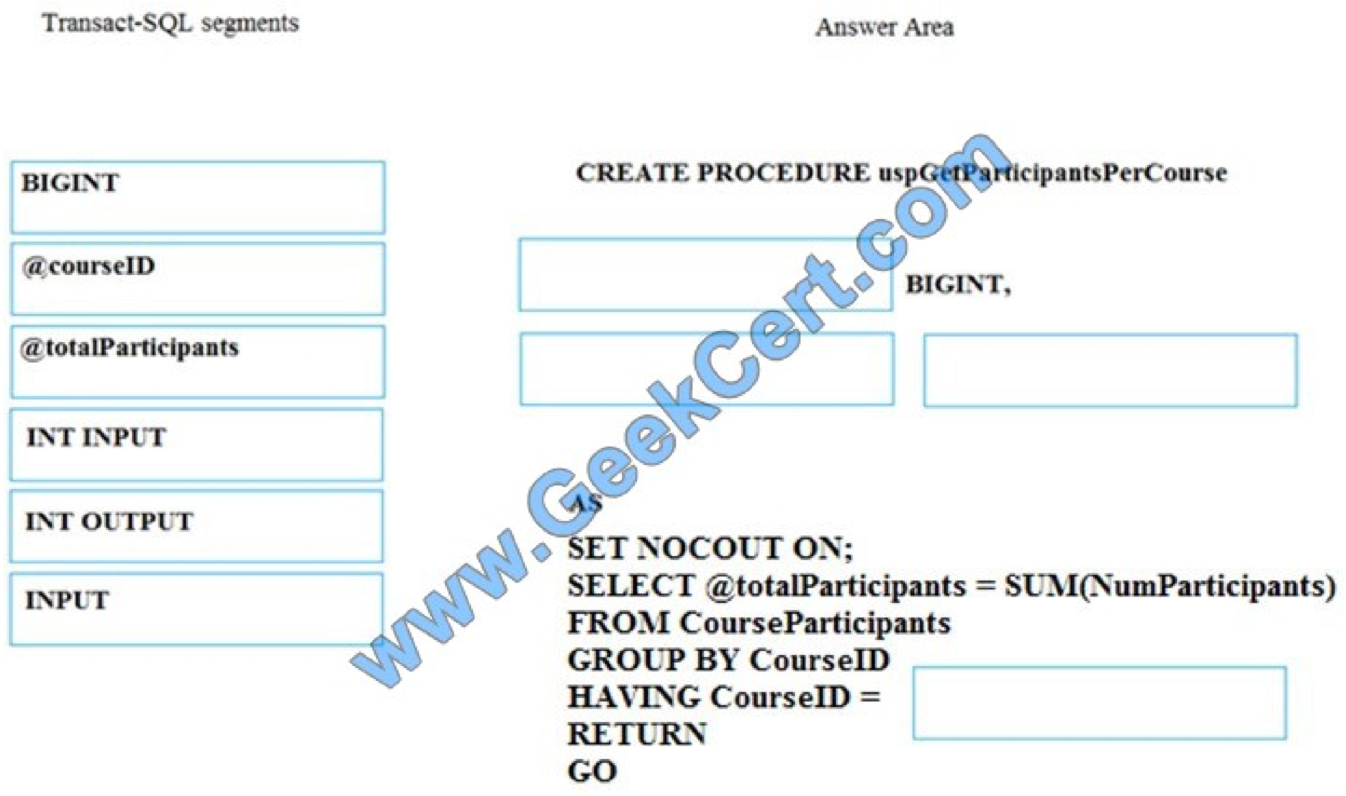
You need to create a stored procedure that returns the total number of participants for a specific course.

How should you complete the procedure? To answer, drag the appropriate Transact-SQL segments to the correct locations. Each Transact-SQL segment may be used once, more than once, or not at all. You may need to drag the split bar

between panes or scroll to view content.

NOTE: Each correct selection is worth one point.

Select and Place:



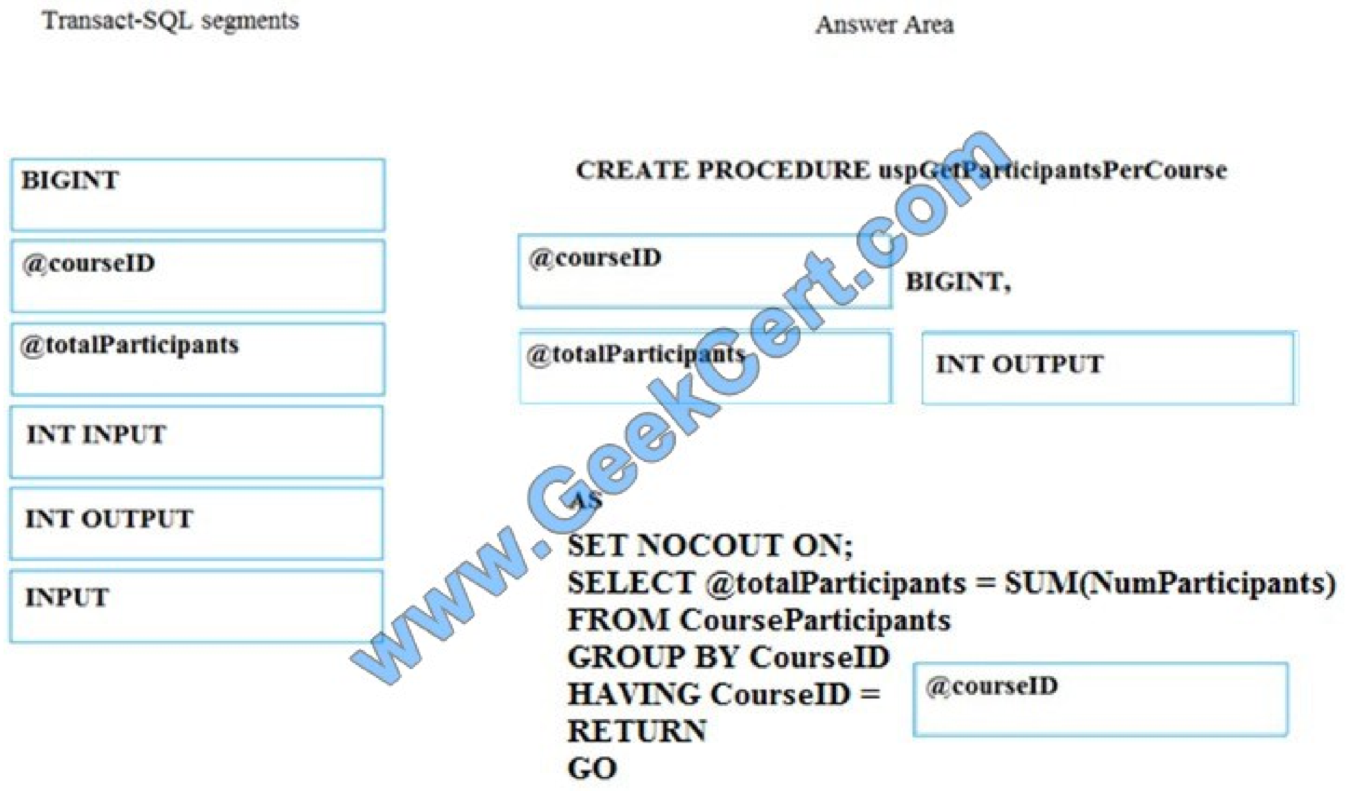
Transact-SQL segments

- BIGINT
- @courseID
- @totalParticipants
- INT INPUT
- INT OUTPUT
- INPUT

Answer Area

```
CREATE PROCEDURE uspGetParticipantsPerCourse
[          ] BIGINT,
[          ] [          ]
AS
SET NOCOUT ON;
SELECT @totalParticipants = SUM(NumParticipants)
FROM CourseParticipants
GROUP BY CourseID
HAVING CourseID = [          ]
RETURN
GO
```

Correct Answer:



Transact-SQL segments

- BIGINT
- @courseID
- @totalParticipants
- INT INPUT
- INT OUTPUT
- INPUT

Answer Area

```
CREATE PROCEDURE uspGetParticipantsPerCourse
@courseID BIGINT,
@totalParticipants INT OUTPUT
AS
SET NOCOUT ON;
SELECT @totalParticipants = SUM(NumParticipants)
FROM CourseParticipants
GROUP BY CourseID
HAVING CourseID = @courseID
RETURN
GO
```

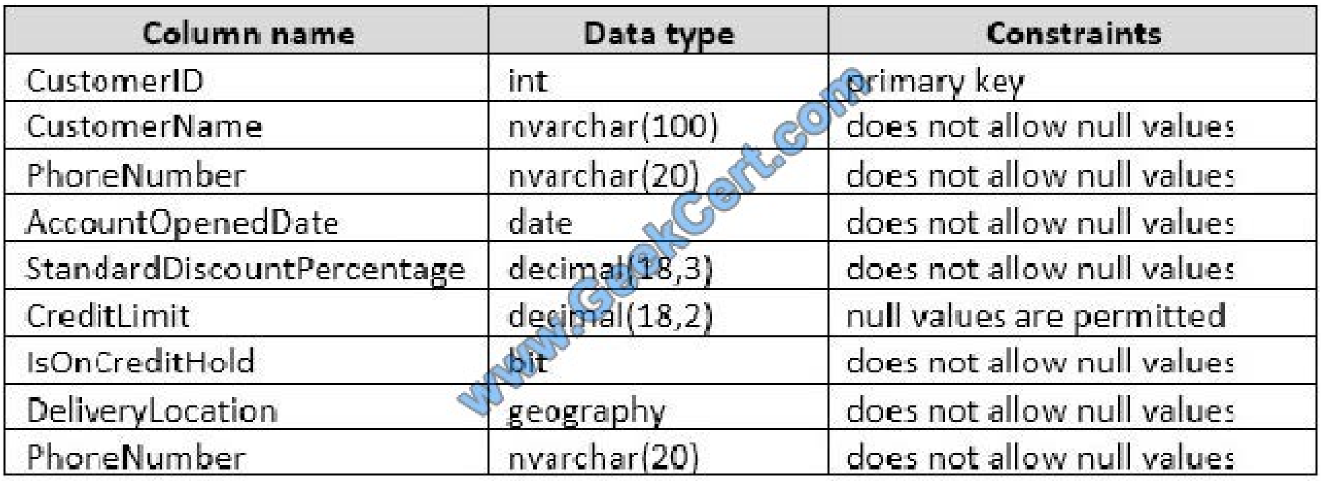
QUESTION 5

DRAG DROP

Note: This question is part of a series of questions that use the same scenario. For your convenience, the scenario is repeated in each question. Each question presents a different goal and answer choices, but the text of the scenario is

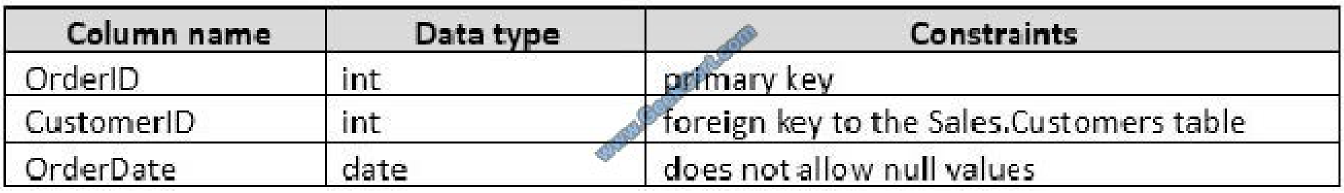
exactly the same in each question in this series.

You are developing a database to track customer orders. The database contains the following tables: Sales.Customers, Sales.Orders, and Sales.OrderLines. The following table describes the columns in Sales.Customers.



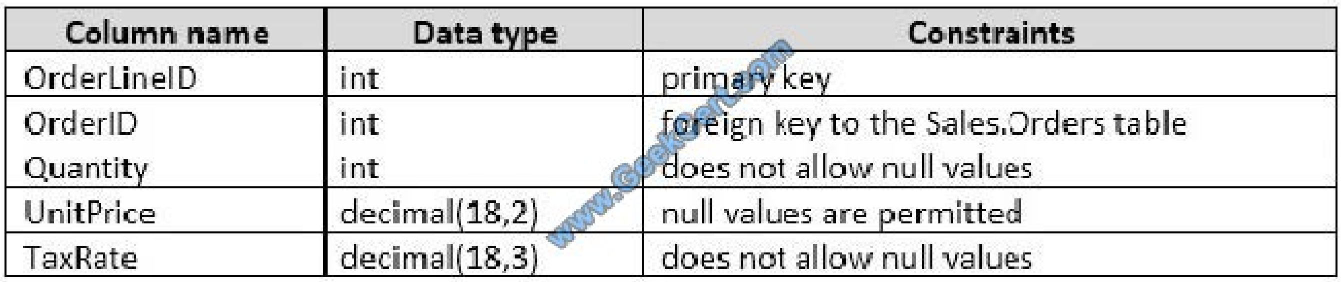
Column name	Data type	Constraints
CustomerID	int	primary key
CustomerName	nvarchar(100)	does not allow null values
PhoneNumber	nvarchar(20)	does not allow null values
AccountOpenedDate	date	does not allow null values
StandardDiscountPercentage	decimal(18,3)	does not allow null values
CreditLimit	decimal(18,2)	null values are permitted
IsOnCreditHold	bit	does not allow null values
DeliveryLocation	geography	does not allow null values
PhoneNumber	nvarchar(20)	does not allow null values

The following table describes the columns in Sales.Orders.



Column name	Data type	Constraints
OrderID	int	primary key
CustomerID	int	foreign key to the Sales.Customers table
OrderDate	date	does not allow null values

The following table describes the columns in Sales.OrderLines.



Column name	Data type	Constraints
OrderLineID	int	primary key
OrderID Quantity	int int	foreign key to the Sales.Orders table does not allow null values
UnitPrice	decimal(18,2)	null values are permitted
TaxRate	decimal(18,3)	does not allow null values

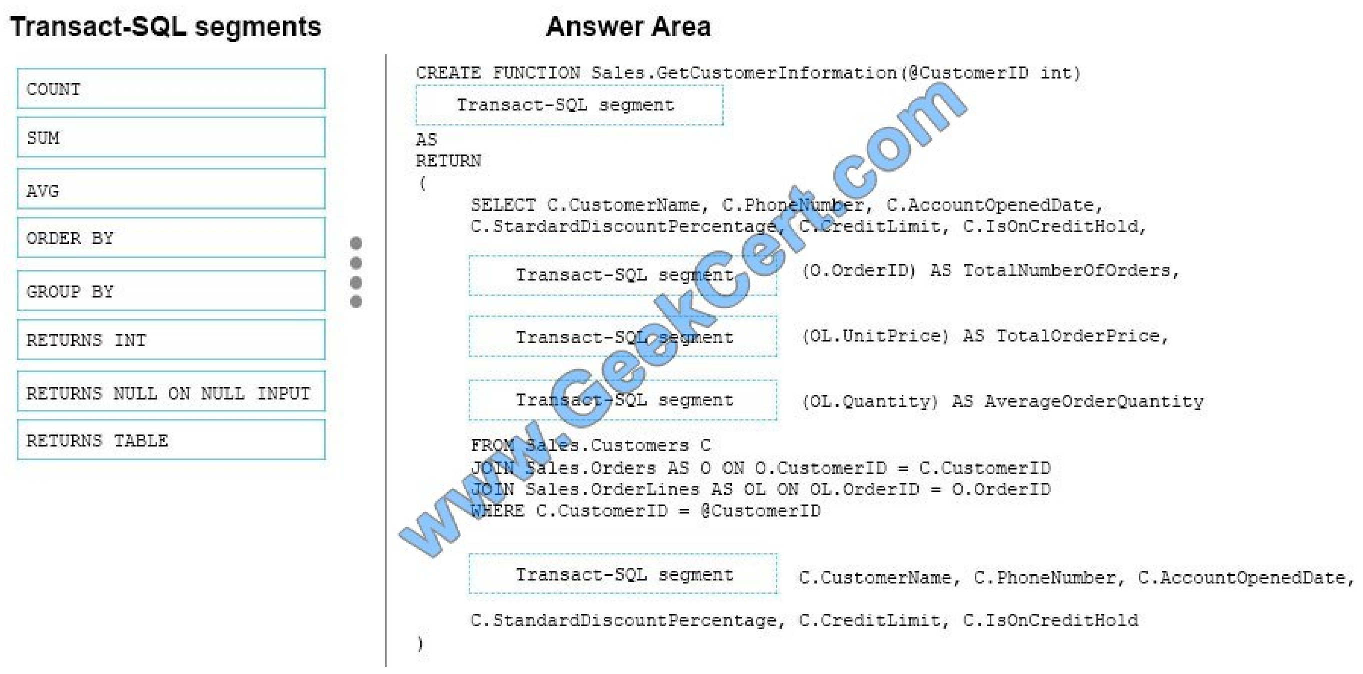
You need to create a function that accepts a CustomerID as a parameter and returns the following information:

all customer information for the customer the total number of orders for the customer

the total price of all orders for the customer

the average quantity of items per order How should you complete the function definition? To answer, drag the appropriate Transact-SQL segment to the correct locations. Transact-SQL segment may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.

Select and Place:



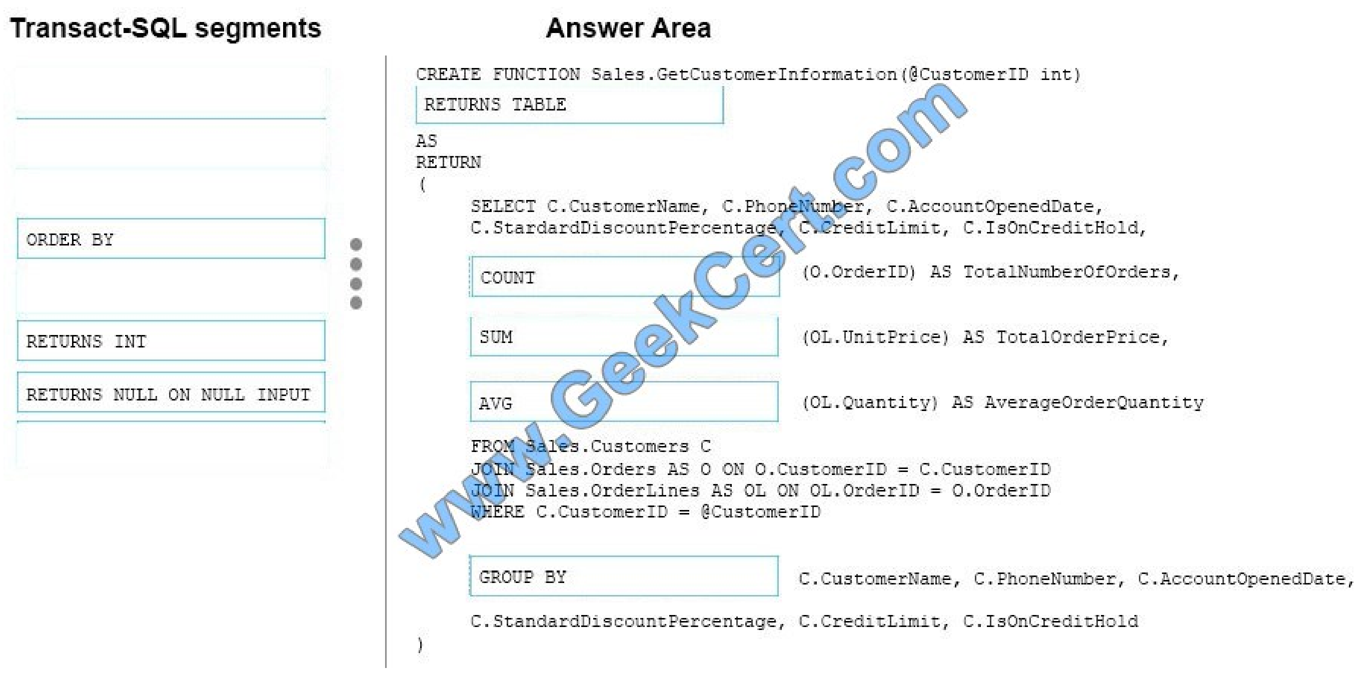
Transact-SQL segments

```
COUNT
SUM
AVG
ORDER BY
GROUP BY
RETURNS INT
RETURNS NULL ON NULL INPUT
RETURNS TABLE
```

Answer Area

```
CREATE FUNCTION Sales.GetCustomerInformation(@CustomerID int)
[Transact-SQL segment]
AS
RETURN
(
     SELECT C.CustomerName, C.PhoneNumber, C.AccountOpenedDate,
     C.StardardDiscountPercentage, C.CreditLimit, C.IsOnCreditHold,
     [Transact-SQL segment] (O.OrderID) AS TotalNumberOfOrders,
     [Transact-SQL segment] (OL.UnitPrice) AS TotalOrderPrice,
     [Transact-SQL segment] (OL.Quantity) AS AverageOrderQuantity
     FROM Sales.Customers C
     JOIN Sales.Orders AS O ON O.CustomerID = C.CustomerID
     JOIN Sales.OrderLines AS OL ON OL.OrderID = O.OrderID
     WHERE C.CustomerID = @CustomerID
     [Transact-SQL segment] C.CustomerName, C.PhoneNumber, C.AccountOpenedDate,
     C.StandardDiscountPercentage, C.CreditLimit, C.IsOnCreditHold
)
```

Correct Answer:



Transact-SQL segments

```
ORDER BY
RETURNS INT
RETURNS NULL ON NULL INPUT
```

Answer Area

```
CREATE FUNCTION Sales.GetCustomerInformation(@CustomerID int)
RETURNS TABLE
AS
RETURN
(
     SELECT C.CustomerName, C.PhoneNumber, C.AccountOpenedDate,
     C.StardardDiscountPercentage, C.CreditLimit, C.IsOnCreditHold,
     COUNT (O.OrderID) AS TotalNumberOfOrders,
     SUM (OL.UnitPrice) AS TotalOrderPrice,
     AVG (OL.Quantity) AS AverageOrderQuantity
     FROM Sales.Customers C
     JOIN Sales.Orders AS O ON O.CustomerID = C.CustomerID
     JOIN Sales.OrderLines AS OL ON OL.OrderID = O.OrderID
     WHERE C.CustomerID = @CustomerID
     GROUP BY C.CustomerName, C.PhoneNumber, C.AccountOpenedDate,
     C.StandardDiscountPercentage, C.CreditLimit, C.IsOnCreditHold
)
```

Box1: RETURNS TABLE

The function should return the following information:

all customer information for the customer

the total number of orders for the customer

the total price of all orders for the customer

the average quantity of items per order

Box 2: COUNT

The function should return the total number of orders for the customer.

Box 3: SUM

The function should return the total price of all orders for the customer.

Box 3. AVG

The function should return the average quantity of items per order.

Box 4: GROUP BY

Need to use GROUP BY for the aggregate functions.

References: https://msdn.microsoft.com/en-us/library/ms186755.aspx