

**100%** Money Back  
**Guarantee**

**Vendor:**Microsoft

**Exam Code:**70-761

**Exam Name:**Querying Data with Transact-SQL

**Version:**Demo

## QUESTION 1

### DRAG DROP

You create three tables by running the following Transact-SQL statements:

```
CREATE TABLE tblRoles (  
    RoleId int NOT NULL IDENTITY(1,1) PRIMARY KEY CLUSTERED,  
    RoleName varchar(20) NOT NULL  
)  
CREATE TABLE tblUsers (  
    UserId int NOT NULL IDENTITY(1,1) PRIMARY KEY CLUSTERED,  
    UserName varchar(20) UNIQUE NOT NULL,  
    IsActive bit NOT NULL DEFAULT(1)  
)  
CREATE TABLE tblUsersInRoles (  
    UserId int NOT NULL FOREIGN KEY REFERENCES tblUsers(UserId),  
    RoleId int NOT NULL FOREIGN KEY REFERENCES tblRoles(RolesId)  
)
```

For reporting purposes, you need to find the active user count for each role, and the total active user count. The result must be ordered by active user count of each role. You must use common table expressions (CTEs). 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

```
Total AS (  
    SELECT COUNT(*) AS TotalCountInAllRoles  
    FROM ActiveUsers  
)  
SELECT S.*, Total.TotalCountInAllRoles  
FROM RoleSummary S, Total  
ORDER BY S.ActiveUserCount
```

```
WITH ActiveUsers AS (  
    SELECT UserId  
    FROM tblUsers  
    WHERE IsActive=1  
)
```

```
RoleNCount AS (  
    SELECT RoleId, COUNT(*) AS ActiveUser-  
Count  
    FROM tblUsersInRoles BRG  
    INNER JOIN ActiveUsers U ON BRG.UserId =  
U.UserId  
    GROUP BY BRG.RoleId  
)
```

```
Total AS (  
    SELECT COUNT(*) AS TotalCountInAllRoles  
    FROM ActiveUsers  
)  
SELECT S.*, Total.TotalCountInAllRoles  
FROM RoleSummary S, Total
```

```
RoleSummary AS (  
    SELECT R.RoleName, ISNULL  
(S.ActiveUserCount,0) AS ActiveUserCount  
    FROM tblRoles R  
    LEFT JOIN RoleNCount S ON R.RoleId =  
S.RoleId  
    ORDER BY S.ActiveUserCount  
)
```

```
RoleSummary AS (  
    SELECT R.RoleName, ISNULL  
(S.ActiveUserCount,0) AS ActiveUserCount  
    FROM tblRoles R  
    LEFT JOIN RoleNCount S ON R.RoleId =  
S.RoleId  
)
```

## Answer Area



Correct Answer:

## Transact-SQL segments

```
Total AS (
    SELECT COUNT(*) AS TotalCountInAllRoles
    FROM ActiveUsers
)
SELECT S.*, Total.TotalCountInAllRoles
FROM RoleSummary S, Total
ORDER BY S.ActiveUserCount
```

```
WITH ActiveUsers AS (
    SELECT UserId
    FROM tblUsers
    WHERE IsActive=1
),
```

```
RoleNCount AS (
    SELECT RoleId, COUNT(*) AS ActiveUser-
    Count
    FROM tblUsersInRoles BRG
    INNER JOIN ActiveUsers U ON BRG.UserId =
    U.UserId
    GROUP BY BRG.RoleId
),
```

```
Total AS (
    SELECT COUNT(*) AS TotalCountInAllRoles
    FROM ActiveUsers
)
SELECT S.*, Total.TotalCountInAllRoles
FROM RoleSummary S, Total
```

```
RoleSummary AS (
    SELECT R.RoleName, ISNULL
    (S.ActiveUserCount,0) AS ActiveUserCount
    FROM tblRoles R
    LEFT JOIN RoleNCount S ON R.RoleId =
    S.RoleId
    ORDER BY S.ActiveUserCount
),
```

```
RoleSummary AS (
    SELECT R.RoleName, ISNULL
    (S.ActiveUserCount,0) AS ActiveUserCount
    FROM tblRoles R
    LEFT JOIN RoleNCount S ON R.RoleId =
    S.RoleId
),
```

## Answer Area

```
RoleNCount AS (
    SELECT RoleId, COUNT(*) AS ActiveUser-
    Count
    FROM tblUsersInRoles BRG
    INNER JOIN ActiveUsers U ON BRG.UserId =
    U.UserId
    GROUP BY BRG.RoleId
),
```

```
WITH ActiveUsers AS (
    SELECT UserId
    FROM tblUsers
    WHERE IsActive=1
),
```

```
RoleSummary AS (
    SELECT R.RoleName, ISNULL
    (S.ActiveUserCount,0) AS ActiveUserCount
    FROM tblRoles R
    LEFT JOIN RoleNCount S ON R.RoleId =
    S.RoleId
    ORDER BY S.ActiveUserCount
),
```

```
Total AS (
    SELECT COUNT(*) AS TotalCountInAllRoles
    FROM ActiveUsers
)
SELECT S.*, Total.TotalCountInAllRoles
FROM RoleSummary S, Total
ORDER BY S.ActiveUserCount
```






## QUESTION 2

### DRAG DROP

You need to create a stored procedure to update a table named Sales.Customers. The structure of the table is shown in the exhibit. (Click the exhibit button.)

## Sales.Customers

### Columns

-  **custid (PK, int, not null)**
-  **companyname (nvarchar(40), not null)**
-  **contactname (nvarchar(30), not null)**
-  **contacttitle (nvarchar(30), not null)**
-  **address(nvarchar(60), not null)**
-  **city(nvarchar(15), not null)**
-  **region(nvarchar(15), null)**
-  **postalcode (nvarchar(10), null)**
-  **country (nvarchar(15), not null)**
-  **phone (nvarchar(24), not null)**
-  **fax (nvarchar(24), null)**

The stored procedure must meet the following requirements:

Accept two input parameters.

Update the company name if the customer exists.

Return a custom error message if the customer does not exist.

Which five 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.

NOTE: More than one order of answer choices is correct. You will receive credit for any of the correct orders you select.

Select and Place:



### Transact-SQL segments

```
CREATE PROCEDURE Sales.ModCompanyName  
@custID int, @newname nvarchar(40) AS  
  
IF NOT EXISTS (SELECT custid FROM  
Sales.Customers WHERE custid = @custID)  
  
UPDATE Sales.Customers  
SET companyname = @newname  
WHERE custid = @custID  
  
BEGIN THROW 55555, 'The customer ID  
does not exist', 1 END  
  
UPDATE Sales.Customers  
SET companyname = @custID  
WHERE custid = @newname  
  
IF EXISTS (SELECT custid FROM  
Sales.Customers  
WHERE custid = @custID)  
  
ROLLBACK TRANSACTION
```

### Answer Area



Correct Answer:

### Transact-SQL segments

```
UPDATE Sales.Customers  
SET companyname = @custID  
WHERE custid = @newname  
  
ROLLBACK TRANSACTION
```

### Answer Area

```
CREATE PROCEDURE Sales.ModCompanyName  
@custID int, @newname nvarchar(40) AS  
  
IF EXISTS (SELECT custid FROM  
Sales.Customers  
WHERE custid = @custID)  
  
UPDATE Sales.Customers  
SET companyname = @newname  
WHERE custid = @custID  
  
IF NOT EXISTS (SELECT custid FROM  
Sales.Customers WHERE custid = @custID)  
  
BEGIN THROW 55555, 'The customer ID  
does not exist', 1 END
```



### QUESTION 3

You are developing a mobile app to manage meetups. The app allows for users to view the 25 closest people with similar interests. You have a table that contains records for approximately two million people. You create the table by running the following Transact-SQL statement:

```
CREATE TABLE Person (
    PersonID INT,
    Name NVARCHAR(155) NOT NULL,
    Location GEOGRAPHY,
    Interests NVARCHAR(MAX)
)
```

You create the following table valued function to generate lists of people:

```
CREATE FUNCTION dbo.nearby (@person AS INT)
    RETURNS @Res TABLE (
        PersonId INT NOT NULL,
        Location GEOGRAPHY
    )
    AS
    BEGIN
        . . .
    END
```

You need to build a report that shows meetings with at least two people only. What should you use?

- A. OUTER APPLY
- B. CROSS APPLY
- C. PIVOT
- D. LEFT OUTER JOIN

Correct Answer: B

References: <https://www.sqlshack.com/the-difference-between-cross-apply-and-outer-apply-in-sql-server/>

---

#### QUESTION 4

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while

others might not have a correct solution.

After you answer a question in this section. You will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You create a table named Products by running the following Transact-SQL statement:

```

CREATE TABLE Products (
    ProductID int IDENTITY (1, 1), NOT NULL PRIMARY KEY,
    ProductName nvarchar (100), NULL,
    UnitPrice decimal (18, 2) NOT NULL,
    UnitsInStock int NOT NULL,
    UnitsOnOrder int NULL
)

```

You have the following stored procedure:

```

CREATE PROCEDURE InsertProduct
    @ProductName nvarchar(100),
    @UnitPrice decimal (18, 2),
    @UnitsInStock int,
    @UnitsOnOrder int
AS
BEGIN
    INSERT INTO Products (ProductName, UnitPrice, UnitsInStock, UnitsOnOrder)
    VALUES (@ProductName, @UnitPrice, @UnitsInStock, @UnitsOnOrder)
END

```

You need to modify the stored procedure to meet the following new requirements:

Insert product records as a single unit of work.

Return error number 51000 when a product fails to insert into the database.

If a product record insert operation fails, the product information must not be permanently written to the database.

Solution: You run the following Transact-SQL statement:



```

ALTER PROCEDURE InsertProduct
@ProductName nvarchar (100),
@UnitPrice decimal (18, 2),
@UnitsInStock int,
@UnitsOnOrder int
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
            INSERT INTO Products (ProductName, UnitPrice, UnitsInStock, UnitsOnOrder)
            VALUES (@ProductName, @UnitPrice, @UnitsInStock, @UnitsOnOrder)
        COMMIT TRANSACTION
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION
        RAISERROR (51000,16, 1)
    END CATCH
END

```

Does the solution meet the goal?

A. Yes

B. No

Correct Answer: B

#### QUESTION 5

You need to create a database object that meets the following requirements:

accepts a product identifies as input

calculates the total quantity of a specific product, including quantity on hand and quantity on order

caches and reuses execution plan

returns a value

can be called from within a SELECT statement

can be used in a JOIN clause

What should you create?

- A. an extended stored procedure
- B. a user-defined table-valued function
- C. a user-defined stored procedure that has an OUTPUT parameter
- D. a memory-optimized table that has updated statistics

Correct Answer: B

References: <https://www.techrepublic.com/blog/the-enterprise-cloud/understand-when-to-use-user-defined-functions-in-sql-server/>

---

## QUESTION 6

### HOTSPOT

You have a database that contains the following tables: tblRoles, tblUsers, and tblUsersInRoles.

The table tblRoles is defined as follows.

| Column name | Data type   | Nullable | Primary key |
|-------------|-------------|----------|-------------|
| RoleID      | int         | No       | Yes         |
| RoleName    | varchar(20) | No       | No          |

You have a function named ufnGetRoleActiveUsers that was created by running the following Transact-SQL statement:

```
CREATE FUNCTION ufnGetRoleActiveUsers(@RoleId AS int)
RETURNS @roleSummary TABLE(UserName varchar (20))
AS
BEGIN
    INSERT INTO @roleSummary
    SELECT U.UserName FROM tblUsersInRoles BRG
    INNER JOIN tblUsers U
    ON U.UserId = BRG.UserId
    WHERE BRG.RoleId = @RoleId AND U.IsActive = 1
    RETURN
END
```

You need to list all roles and their corresponding active users. The query must return the RoleId, RoleName, and UserName columns. If a role has no active users, a NULL value should be returned as the UserName for that role. How should you complete the Transact-SQL statement? To answer, select the appropriate Transact-SQL segments in the answer area.

Hot Area:

### Answer area

SELECT \*

FROM

|                 |
|-----------------|
|                 |
| tblRoles        |
| tblUsersInRoles |
| tblUsers        |

|             |   |
|-------------|---|
|             | ▼ |
| CROSS JOIN  |   |
| OUTER APPLY |   |
| CROSS APPLY |   |

ufnGetRoleActiveUsers(

|          |   |   |
|----------|---|---|
|          | ▼ | ) |
| RoleId   |   |   |
| UserId   |   |   |
| RoleName |   |   |

Correct Answer:

### Answer area

SELECT \*

FROM

|                 |
|-----------------|
|                 |
| tblRoles        |
| tblUsersInRoles |
| tblUsers        |

|             |   |
|-------------|---|
|             | ▼ |
| CROSS JOIN  |   |
| OUTER APPLY |   |
| CROSS APPLY |   |

ufnGetRoleActiveUsers(

|          |   |   |
|----------|---|---|
|          | ▼ | ) |
| RoleId   |   |   |
| UserId   |   |   |
| RoleName |   |   |

### QUESTION 7

You have a database that stores information about server and application errors. The database contains the following table: Servers

| Column   | Data type     | Notes  |
|----------|---------------|--|
| ServerID | int           | This is the primary key for the table.         |
| DNS      | Nvarchar(100) | Null values are not permitted for this column. |

Errors

| Column      | Data type     | Notes   |
|-------------|---------------|---|
| ErrorID     | int           | This is the primary key for the table.  |
| ServerID    | int           | Null values are not permitted for this column. This column is a foreign key that is related for the <code>ServerID</code> column in the <code>Servers</code> table. |
| Occurrences | int           | Null values are not permitted for this column.  |
| LogMessage  | nvarchar(max) | Null values are not permitted for this column.  |

You need to return all unique error log messages and the server where the error occurs most often. Which Transact-SQL statement should you run?

```
SELECT DISTINCT ServerID, LogMessage FROM Errors AS e1
WHERE LogMessage IN (
    SELECT TOP 1 e2.LogMessage FROM Errors AS e2
    WHERE e2.LogMessage = e1.LogMessage AND e2.ServerID <> e1.ServerID
    ORDER BY e2.Occurrences
)
```

- A. 

```
SELECT DISTINCT ServerID, LogMessage FROM Errors AS e1
WHERE LogMessage IN (
    SELECT TOP 1 e2.LogMessage FROM Errors AS e2
    WHERE e2.LogMessage = e1.LogMessage AND e2.ServerID <> e1.ServerID
    ORDER BY e2.Occurrences
)
```
- B. 

```
SELECT DISTINCT ServerID, LogMessage FROM Errors AS e1
WHERE Occurrences > ALL (
    SELECT e2.LogMessage FROM Errors AS e2
    WHERE e2.LogMessage = e1.LogMessage AND e2.ServerID <> e1.ServerID
)
```
- C. 

```
SELECT DISTINCT ServerID, LogMessage FROM Errors AS e1
GROUP BY ServerID, LogMessage
HAVING MAX(Occurrences) = 1
```
- D. 

```
SELECT ServerID, LogMessage FROM Errors AS e1
GROUP BY ServerID, LogMessage, Occurrences
HAVING COUNT(*) = 1
ORDER BY Occurrences
```

A. B. C. D.

Correct Answer: A

## QUESTION 8

Note: This question is part of a series of questions that use the same or similar answer choices. An answer choice may be correct for more than one question in the series. Each question is independent of the other questions in this series. Information and details provided in a question apply to that question.

You have a database for a banking system. The database has two tables named tblDepositAcct and tblLoanAcct that store deposit and loan accounts, respectively. Both tables contain the following columns:

| Column name | Data type  | Primary key column | Description   |
|-------------|------------|--------------------|---|
| CustNo      | int        | No                 | This column uniquely identifies a customer in the bank. A customer may have both deposit and loan accounts.             |
| AcctNo      | int        | Yes                | This column uniquely identifies a customer in the bank.   |
| ProdCode    | varchar(3) | No                 | This column identifies the product type of an account. A customer may have multiple accounts for the same product type. |

You need to run a query to find the total number of customers who have both deposit and loan accounts. Which Transact-SQL statement should you run?

```
SELECT COUNT(*)
FROM (SELECT AcctNo
      FROM tblDepositAcct
      INTER
      SECTSELECT Acct
      NoFROM tblLoanAcct) R
```



- A. `SELECT COUNT(*)`  
    `FROM (SELECT AcctNo`  
        `FROM tblDepositAcct`  
        `INTER`  
        `SECTSELECT Acct`  
        `NoFROM tblLoanAcct) R`
- B. `SELECT COUNT(*)`  
    `FROM (SELECT CustNo`  
        `FROM tblDepositAcct`  
        `UNION`  
        `SELECT CustNo`  
        `FROM tblLoanAcct) R`
- C. `SELECT COUNT(*)`  
    `FROM (SELECT CustNo`  
        `FROM tblDepositAcct`  
        `UNION ALL`  
        `SELECT CustNo`  
        `FROM tblLoanAcct) R`
- D. `SELECT COUNT (DISTINCT D.CustNo)`  
    `FROM tblDepositAcct D, tblLoanAcct L`  
    `WHERE D.CustNo = L.CustNo`
- E. `SELECT COUNT(DISTINCT L.CustNo)`  
    `FROM tblDepositAcct D`  
    `RIGHT JOIN tblLoanAcct L ON D.CustNo = L.CustNo`  
    `WHERE D.CustNo IS NULL`
- F. `SELECT COUNT(*)`  
    `FROM (SELECT CustNo`  
        `FROM tblDepositAcct`  
        `EXCEPT`  
        `SELECT CustNo`  
        `FROM tblLoanAcct) R`
- G. `SELECT COUNT (DISTINCT COALESCE(D.CustNo, L.CustNo))`  
    `FROM tblDepositAcct D`  
    `FULL JOIN tblLoanAcct L ON D.CustNo = L.CustNo`  
    `WHERE D.CustNo IS NULL OR L.CustNo IS NULL`
- H. `SELECT COUNT(*)`  
    `FROM tblDepositAcct D`  
    `FULL JOIN tblLoanAcct L ON D.CustNo = L.CustNo`



A. B. C. D. E. F. G. H.

Correct Answer: A

The SQL INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.

References: <https://www.techonthenet.com/sql/intersect.php>

---

## QUESTION 9

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while

others might not have a correct solution.

After you answer a question in this section. You will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You have a database that contains a single table named tblVehicleRegistration. The table is defined as follows:

| Column name        | Data type  | Description  |
|--------------------|------------|--|
| VehicleId          | int        | the primary key for the table  |
| RegistrationNumber | varchar(5) | a vehicle registration number that contains only letters and numbers |
| RegistrationDate   | date       | the vehicle registration date  |
| UserId             | int        | an identifier for the vehicle owner                                  |

You run the following query:

```
SELECT UserId FROM tblVehicleRegistration
WHERE RegistrationNumber = 20012
AND RegistrationDate > '2016-01-01'
```

The query output window displays the following error message: "Conversion failed when converting the varchar value 'AB012\' to data type int." You need to resolve the error. Solution: You modify the Transact-SQL statement as follows:

```
SELECT UserId FROM tblVehicleRegistration
WHERE RegistrationNumber = 20012
AND RegistrationDate > CONVERT(DATE, '2016-01-01', 120)
```

Does the solution meet the goal?

A. Yes

B. No

Correct Answer: B

---

## QUESTION 10

### 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 on this series.

You have a database that tracks orders and deliveries for customers in North America. System versioning is enabled for all tables. The database contains the Sales.Customers, Application.Cities, and Sales.CustomerCategories tables.

Details for the Sales.Customers table are shown in the following table:

| Column                     | Data type     | Notes  |
|----------------------------|---------------|--|
| CustomerId                 | int           | primary key  |
| CustomerCategoryId         | int           | foreign key to the Sales.CustomerCategories table    |
| PostalCityID               | int           | foreign key to the Application.Cities table          |
| DeliveryCityID             | int           | foreign key to the Application.Cities table          |
| AccountOpenedDate          | datetime      | does not allow values                                |
| StandardDiscountPercentage | int           | does not allow values                                |
| CreditLimit                | decimal(18,2) | null values are permitted                            |
| IsOnCreditHold             | bit           | does not allow values                                |
| DeliveryLocation           | geography     | does not allow values                                |
| PhoneNumber                | nvarchar(20)  | does not allow values                                |
| ValidFrom                  | datetime2(7)  | does not allow values, GENERATED ALWAYS AS ROW START |
| ValidTo                    | datetime2(7)  | does not allow values, GENERATED ALWAYS AS ROW END   |

Details for the Application.Cities table are shown in the following table:

| Column                   | Data type | Notes                     |
|--------------------------|-----------|---------------------------|
| CityID                   | int       | primary key               |
| LatestRecordedPopulation | bigint    | null values are permitted |

Details for the Sales.CustomerCategories table are shown in the following table:

| Column               | Data type    | Notes                      |
|----------------------|--------------|----------------------------|
| CustomerCategoryID   | int          | primary key                |
| CustomerCategoryName | nvarchar(50) | does not allow null values |

The marketing department is performing an analysis of how discount affect credit limits. They need to know the average credit limit per standard discount percentage for customers whose standard discount percentage is between zero and four.

You need to create a query that returns the data for the analysis.

How should you complete the Transact-SQL statement? To answer, drag the appropriate Transact-SQL segments to the correct locations. Each Transact-SQL segments 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

0, 1, 2, 3, 4

(0...4)

BETWEEN 0 AND 4

PIVOT

GROUP BY

[CreditLimit]

AVG(CreditLimit)

### Answer Area

```

SELECT
FROM (
    SELECT
        StandardDiscountPercentage,
    FROM Sales.Customers
) AS SourceTable
(
    AVG(CreditLimit)
    FOR StandardDiscountPercentage IN
) AS CreditLimitTable
  
```

Correct Answer:

### Transact-SQL segments

0, 1, 2, 3, 4

(0...4)

BETWEEN 0 AND 4

PIVOT

GROUP BY

[CreditLimit]

AVG(CreditLimit)

### Answer Area

```

SELECT
FROM (
    SELECT
        StandardDiscountPercentage,
    FROM Sales.Customers
) AS SourceTable
PIVOT
(
    AVG(CreditLimit)
    FOR StandardDiscountPercentage IN
) AS CreditLimitTable
  
```

Box 1: 0, 1, 2, 3, 4

Pivot example:

-- Pivot table with one row and five columns

```

SELECT \"AverageCost\" AS Cost_Sorted_By_Production_Days,
[0], [1], [2], [3], [4]
FROM
(SELECT DaysToManufacture, StandardCost
FROM Production.Product) AS SourceTable
PIVOT
(
AVG(StandardCost)
FOR DaysToManufacture IN ([0], [1], [2], [3], [4])
) AS PivotTable;

```

Box 2: [CreditLimit]

Box 3: PIVOT

You can use the PIVOT and UNPIVOT relational operators to change a table-valued expression into another table. PIVOT rotates a table-valued expression by turning the unique values from one column in the expression into multiple columns in the output, and performs aggregations where they are required on any remaining column values that are wanted in the final output.

Box 4: 0, 1, 2, 3, 4

The IN clause determines whether a specified value matches any value in a subquery or a list.

Syntax: test\_expression [ NOT ] IN ( subquery | expression [ ,...n ] )

Where expression[ ,... n ] is a list of expressions to test for a match. All expressions must be of the same type as test\_expression.

References: [https://technet.microsoft.com/en-us/library/ms177410\(v=sql.105\).aspx](https://technet.microsoft.com/en-us/library/ms177410(v=sql.105).aspx)

---

## QUESTION 11

### SIMULATION

You have a database that contains the following tables.



You need to create a query that lists the highest-performing salespersons based on the current year-to-date sales period. The query must meet the following requirements:

Return the LastName and SalesYTD for the three salespersons with the highest year-to-date sales values.

Exclude salespersons that have no value for TerritoryID.

Construct the query using the following guidelines:

Use the first letter of a table name as the table alias.

Use two-part column names.

Do not surround object names with square brackets.

Do not use implicit joins.

Use only single quotes for literal text.

Use aliases only if required.

## Keywords

|                   |                 |                                |
|-------------------|-----------------|--------------------------------|
| ADD               | EXIT            | PROC                           |
| ALL               | EXTERNAL        | PROCEDURE                      |
| ALTER             | FETCH           | PUBLIC                         |
| AND               | FILE            | RAISERROR                      |
| ANY               | FILLFACTOR      | READ                           |
| AS                | FORFOREIGN      | READTEXT                       |
| ASC               | FREETEXT        | RECONFIGURE                    |
| AUTHORIZATION     | FREETEXTTABLE   | REFERENCES                     |
| BACKUP            | FROM            | REPLICATION                    |
| BEGIN             | FULL            | RESTORE                        |
| BETWEEN           | FUNCTION        | RESTRICT                       |
| BREAK             | GOTO            | RETURN                         |
| BROWSE            | GRANT           | REVERT                         |
| BULK              | GROUP           | REVOKE                         |
| BY                | HAVING          | RIGHT                          |
| CASCADE           | HOLDLOCK        | ROLLBACK                       |
| CASE              | IDENTITY        | ROWCOUNT                       |
| CHECK             | IDENTITY_INSERT | ROWGUIDCOL                     |
| CHECKPOINT        | IDENTITYCOL     | RULE                           |
| CLOSE             | IF              | SAVE                           |
| CLUSTERED         | IN              | SCHEMA                         |
| COALESCE          | INDEX           | SECURITYAUDIT                  |
| COLLATE           | INNER           | SELECT                         |
| COLUMN            | INSERT          | SEMANTICKEYPHRASETABLE         |
| COMMIT            | INTERSECT       | SEMANTICSIMILARITYDETAILSTABLE |
| COMPUTE           | INTO            | SEMANTICSIMILARITYTABLE        |
| CONCAT            | IS              | SESSION_USER                   |
| CONSTRAINT        | JOIN            | SET                            |
| CONTAINS          | KEY             | SETUSER                        |
| CONTAINSTABLE     | KILL            | SHUTDOWN                       |
| CONTINUE          | LEFT            | SOME                           |
| CONVERT           | LIKE            | STATISTICS                     |
| CREATE            | LINENO          | SYSTEM_USER                    |
| CROSS             | LOAD            | TABLE                          |
| CURRENT           | MERGE           | TABLESAMPLE                    |
| CURRENT_DATE      | NATIONAL        | TEXTSIZE                       |
| CURRENT_TIME      | NOCHECK         | THEN                           |
| CURRENT_TIMESTAMP | NONCLUSTERED    | TO                             |
| CURRENT_USER      | NOT             | TOP                            |
| CURSOR            | NULL            | TRAN                           |
| DATABASE          | NULLIF          | TRANSACTION                    |
| DBCC              | OF              | TRIGGER                        |
| DEALLOCATE        | OFF             | TRUNCATE                       |
| DECLARE           | OFFSETS         | TRY_CONVERT                    |
| DEFAULT           | ON              | TSEQUAL                        |
| DELETE            | OPEN            | UNION                          |
| DENY              | OPENDATASOURCE  | UNIQUE                         |
| DESC              | OPENQUERY       | UNPIVOT                        |
| DISK              | OPENROWSET      | UPDATE                         |
| DISTINCT          | OPENXML         | UPDATETEXT                     |
| DISTRIBUTED       | OPTION          | USE                            |
| DOUBLE            | OR              | USER                           |
| DROP              | ORDER           | VALUES                         |
| DUMP              | OUTER           | VARYING                        |
| ELSE              | OVER            | VIEW                           |
| END               | PERCENT         | WAITFOR                        |
| ERRLVL            | PIVOT           | WHEN                           |
| ESCAPE            | PLAN            | WHERE                          |
| ESCEFT            | PRECISION       | WHILE                          |
| EXEC              | PRIMARY         | WITH                           |
| EXECUTE           | PRINT           | WITHIN GROUP                   |
| EXISTS            |                 | WRITETEXT                      |



Part of the correct Transact-SQL has been provided in the answer area below. Enter the code in the answer area that resolves the problem and meets the stated goals or requirements. You can add code within the code that has been provided as well as below it.

```
1 SELECT top 3 lastname,salesYTD 2 FROM Person AS p INNER JOIN SalesPerson AS s 3 ON p.PersonID =  
s.SalesPersonID 4 WHERE territoryid is null 5 order by salesytd dsec
```

Use the Check Syntax button to verify your work. Any syntax or spelling errors will be reported by line and character position.

Correct Answer: Please see

```
1 SELECT top 3 lastname,salesYTD  
2 FROM Person AS p INNER JOIN SalesPerson AS s  
3 ON p.PersonID = s.SalesPersonID  
4 WHERE territoryid is not null  
5 order by salesytd desc
```

Note:

On line 4 add a not before null.

On line 5 change dsec to desc.

---

## QUESTION 12

You need to create a database object that meets the following requirements:

accepts a product identified as input

calculates the total quantity of a specific product, including quantity on hand and quantity on order

caches and reuses execution plan

returns a value

can be called from within a SELECT statement

can be used in a JOIN clause What should you create?

A.

an extended stored procedure

B.

a user-defined scalar function

C.

a user-defined stored procedure that has an OUTPUT parameter

D.

a temporary table that has a columnstore index

Correct Answer: B

User-defined scalar functions are execution plans that accept parameters, perform an action such as a complex calculation, and returns the result of that action as a value. The return value can either be a single scalar value or a result set.

Furthermore the execution plan is cached and reusable.

User-defined scalar functions can also be called from within a SELECT statement and can be used in a JOIN clause.

Incorrect Answers:

A: Using extended stored procedures is not recommended as they has been deprecated. CLR Integration should be used instead of extended stored procedures.

C: Stored procedures cannot be used in a SELECT statement or in a JOIN clause.

D: A temporary table is a result set and not a value.

References:

<https://www.c-sharpcorner.com/UploadFile/996353/difference-between-stored-procedure-and-user-defined-functio/>

To Read the [Whole Q&As](#), please purchase the [Complete Version](#) from [Our website](#).

## Try our product !

**100%** Guaranteed Success

**100%** Money Back Guarantee

**365** Days Free Update

**Instant Download** After Purchase

**24x7** Customer Support

Average **99.9%** Success Rate

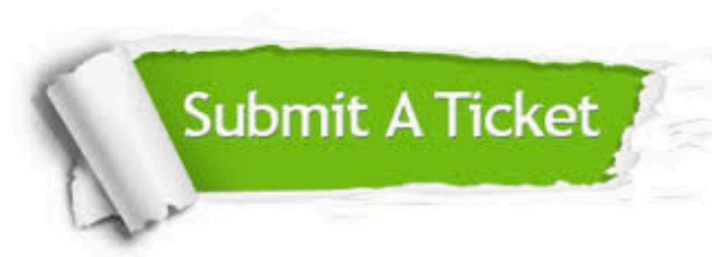
More than **800,000** Satisfied Customers Worldwide

Multi-Platform capabilities - **Windows, Mac, Android, iPhone, iPod, iPad, Kindle**

## Need Help

Please provide as much detail as possible so we can best assist you.

To update a previously submitted ticket:



|   |   |  |
|---|---|--|
|  <b>One Year Free Update</b> <p>Free update is available within One Year after your purchase. After One Year, you will get 50% discounts for updating. And we are proud to boast a 24/7 efficient Customer Support system via Email.</p> |  <b>Money Back Guarantee</b> <p>To ensure that you are spending on quality products, we provide 100% money back guarantee for 30 days from the date of purchase.</p> |  <b>Security &amp; Privacy</b> <p>We respect customer privacy. We use McAfee's security service to provide you with utmost security for your personal information &amp; peace of mind.</p> |
|---|---|--|

Any charges made through this site will appear as Global Simulators Limited.

All trademarks are the property of their respective owners.