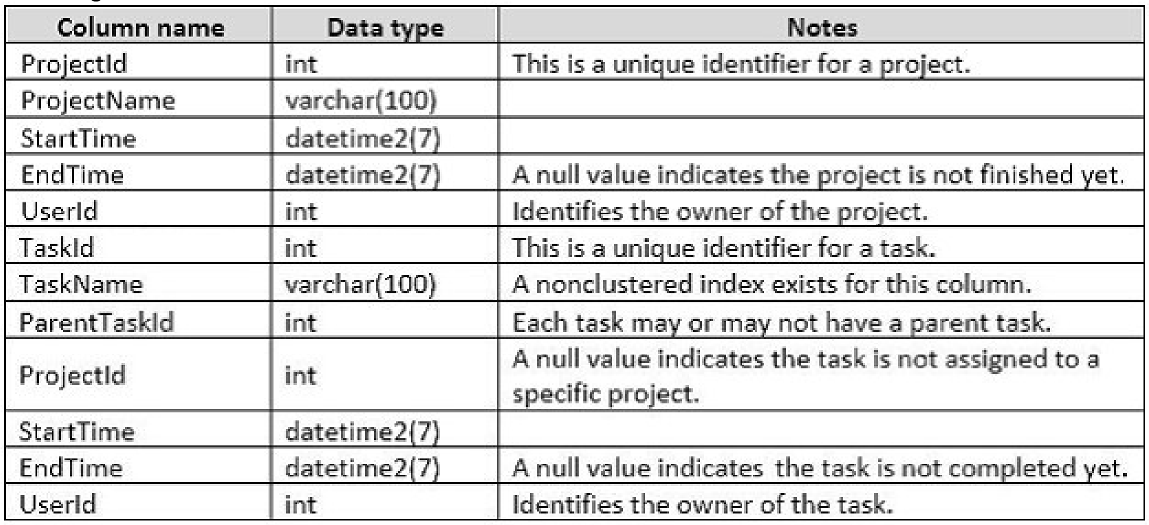
➢ Vendor: Microsoft

➢ Exam Code: 70-761

➢ Exam Name: Querying Data with Transact-SQL

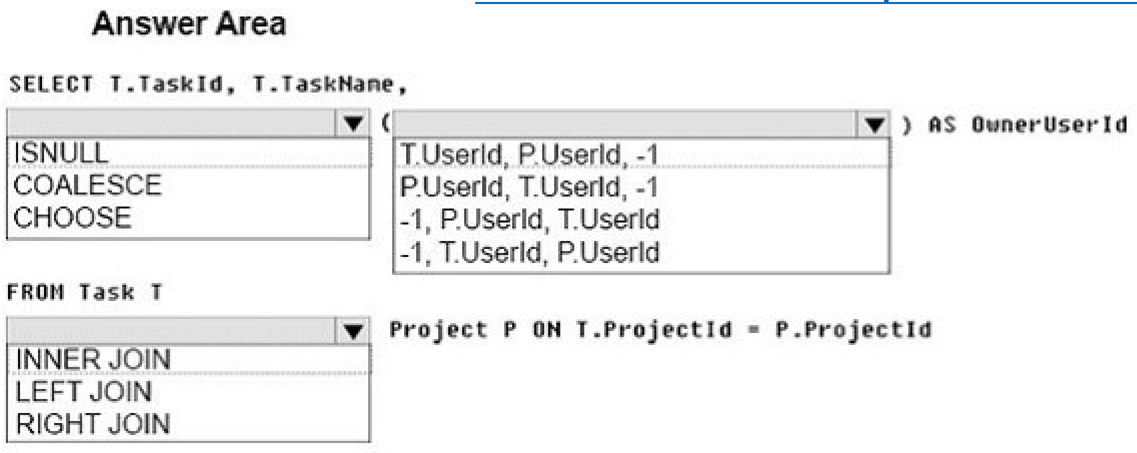
➢ Question 11 – Question 20

QUESTION 11
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.
Hotspot Question
You query a database that includes two tables: Project and Task. The Project table includes the following columns:



Column name	Data type	Notes
ProjectId	int	This is a unique identifier for a project.
ProjectName	varchar(100)	
StartTime	datetime2(7)	
EndTime	datetime2(7)	A null value indicates the project is not finished yet.
UserId	int	Identifies the owner of the project.
TaskId	int	This is a unique identifier for a task.
TaskName	varchar(100)	A nonclustered index exists for this column.
ParentTaskId	int	Each task may or may not have a parent task.
ProjectId	int	A null value indicates the task is not assigned to a specific project.
StartTime	datetime2(7)	
EndTime	datetime2(7)	A null value indicates the task is not completed yet.
UserId	int	Identifies the owner of the task.

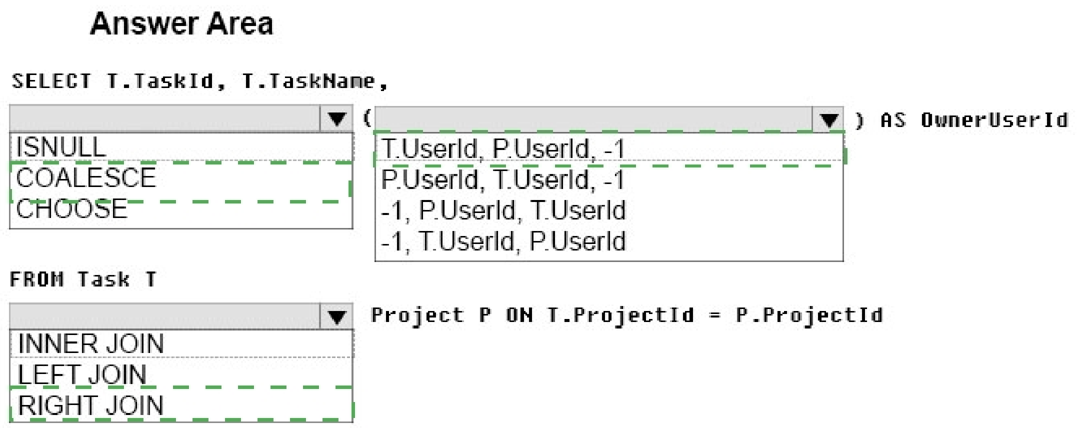
You need to identify the owner of each task by using the following rules:
```
- Return each task's owner if the task has an owner.
- If a task has no owner, but is associated with a project that has an
owner, return the project's owner.
- Return the value -1 for all other cases.
```
How should you complete the Transact-SQL statement? To answer, select the appropriate Transact-SQL segments in the answer area.



Answer Area

```
SELECT T.TaskId, T.TaskName,
[ ▼ ] ( [ ▼ ] ) AS OwnerUserId
ISNULL          T.UserId, P.UserId, -1
COALESCE        P.UserId, T.UserId, -1
CHOOSE          -1, P.UserId, T.UserId
                -1, T.UserId, P.UserId
FROM Task T
[ ▼ ] Project P ON T.ProjectId = P.ProjectId
INNER JOIN
LEFT JOIN
RIGHT JOIN
```

Answer:



Answer Area

```
SELECT T.TaskId, T.TaskName,
[ ▼ ] ( [ ▼ ] ) AS OwnerUserId
ISNULL            [✓ T.UserId, P.UserId, -1]
[✓ COALESCE]      P.UserId, T.UserId, -1
CHOOSE            -1, P.UserId, T.UserId
                  -1, T.UserId, P.UserId
FROM Task T
[ ▼ ] Project P ON T.ProjectId = P.ProjectId
INNER JOIN
LEFT JOIN
[✓ RIGHT JOIN]
```

Explanation:
Box 1: COALESCE
COALESCE evaluates the arguments in order and returns the current value of the first expression that initially does not evaluate to NULL.
Box 2: T.UserID, p.UserID, -1
- Return each task's owner if the task has an owner.
- If a task has no owner, but is associated with a project that has an owner, return the project's owner.
- Return the value -1 for all other cases.
Box 3: RIGHT JOIN
The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match. Here the right side could be NULL as the projectID of the task could be NULL.
References:
https://msdn.microsoft.com/en-us/library/ms190349.aspx
http://www.w3schools.com/Sql/sql_join_right.asp

QUESTION 12
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.
Drag and Drop Question
You query a database that includes two tables: Project and Task. The Project table includes the

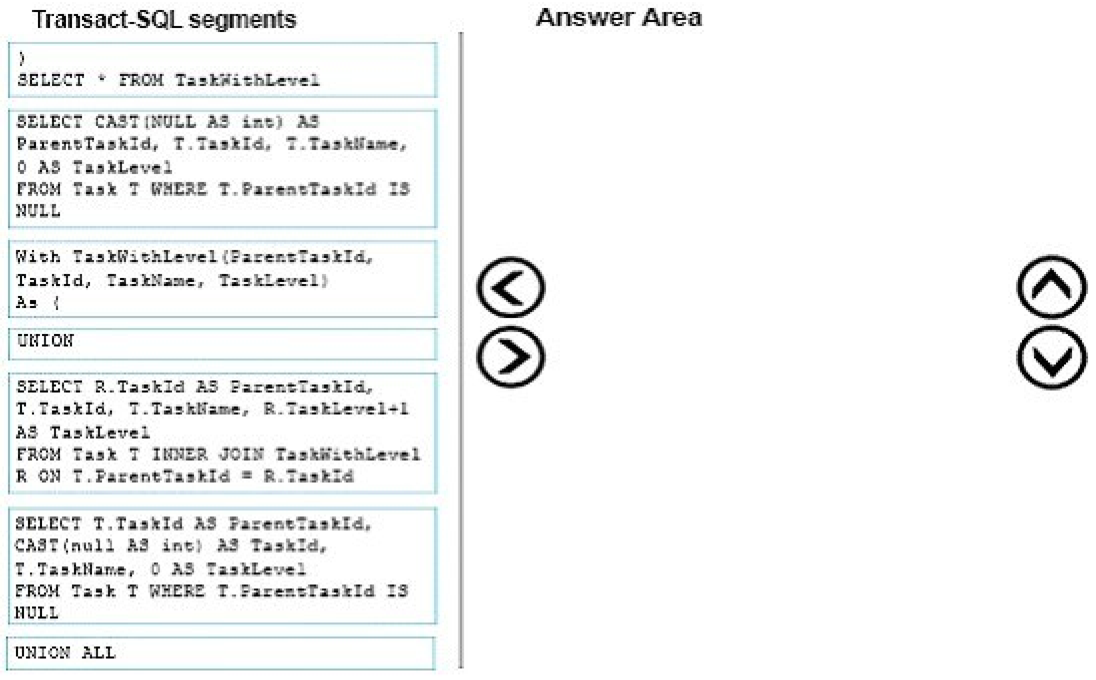
following columns:

Column name	Data type	Notes
ProjectId	int	This is a unique identifier for a project.
ProjectName	varchar(100)	
StartTime	datetime2(7)	
EndTime	datetime2(7)	A null value indicates the project is not finished yet.
UserId	int	Identifies the owner of the project.
TaskId	int	This is a unique identifier for a task.
TaskName	varchar(100)	A nonclustered index exists for this column.
ParentTaskId	int	Each task may or may not have a parent task.
ProjectId	int	A null value indicates the task is not assigned to a specific project.
StartTime	datetime2(7)	
EndTime	datetime2(7)	A null value indicates the task is not completed yet.
UserId	int	Identifies the owner of the task.

Task level is defined using the following rules:

Task category	Task level definition
Tasks that have no parent task	[Task Level] = 0
Tasks that have a parent task	[Task Level] = [Parent Task's Level] + 1

You need to determine the task level for each task in the hierarchy. 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.



Transact-SQL segments

```
)
SELECT * FROM TaskWithLevel
```

```
SELECT CAST(NULL AS int) AS
ParentTaskId, T.TaskId, T.TaskName,
0 AS TaskLevel
FROM Task T WHERE T.ParentTaskId IS
NULL
```

```
With TaskWithLevel(ParentTaskId,
TaskId, TaskName, TaskLevel)
As (
```

```
UNION
```

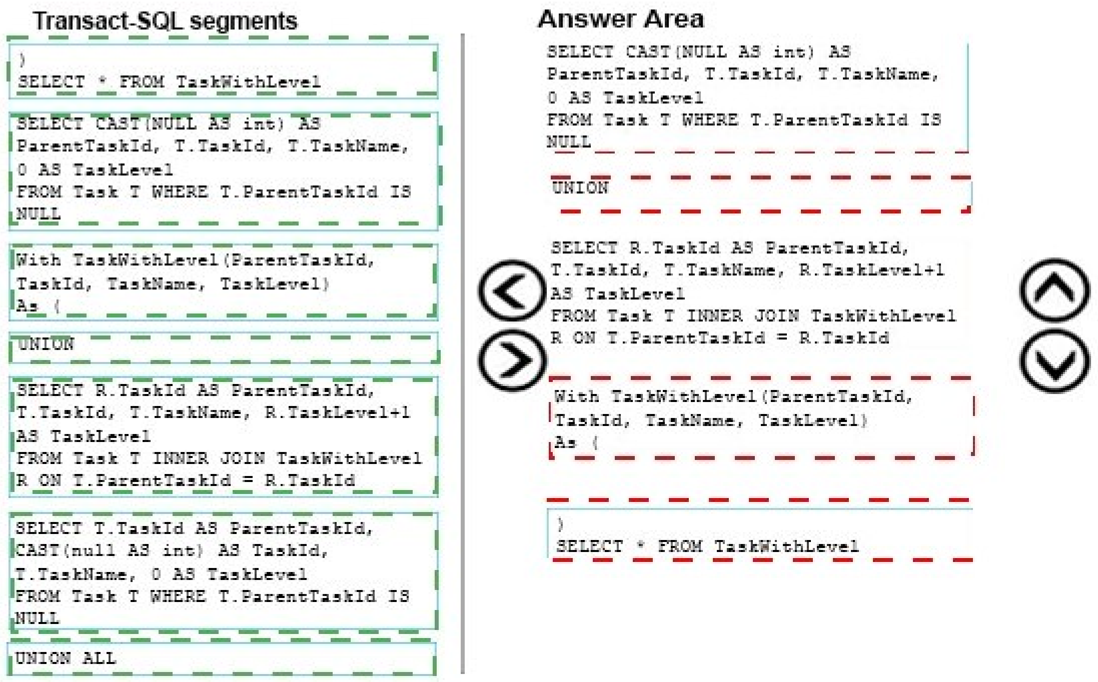
```
SELECT R.TaskId AS ParentTaskId,
T.TaskId, T.TaskName, R.TaskLevel+1
AS TaskLevel
FROM Task T INNER JOIN TaskWithLevel
R ON T.ParentTaskId = R.TaskId
```

```
SELECT T.TaskId AS ParentTaskId,
CAST(null AS int) AS TaskId,
T.TaskName, 0 AS TaskLevel
FROM Task T WHERE T.ParentTaskId IS
NULL
```

```
UNION ALL
```

Answer Area

Answer:



Transact-SQL segments:

```
)
SELECT * FROM TaskWithLevel
```

```
SELECT CAST(NULL AS int) AS
ParentTaskId, T.TaskId, T.TaskName,
0 AS TaskLevel
FROM Task T WHERE T.ParentTaskId IS
NULL
```

```
With TaskWithLevel(ParentTaskId,
TaskId, TaskName, TaskLevel)
As (
```

```
UNION
```

```
SELECT R.TaskId AS ParentTaskId,
T.TaskId, T.TaskName, R.TaskLevel+1
AS TaskLevel
FROM Task T INNER JOIN TaskWithLevel
R ON T.ParentTaskId = R.TaskId
```

```
SELECT T.TaskId AS ParentTaskId,
CAST(null AS int) AS TaskId,
T.TaskName, 0 AS TaskLevel
FROM Task T WHERE T.ParentTaskId IS
NULL
```

```
UNION ALL
```

Answer Area:

```
SELECT CAST(NULL AS int) AS
ParentTaskId, T.TaskId, T.TaskName,
0 AS TaskLevel
FROM Task T WHERE T.ParentTaskId IS
NULL
```

```
UNION
```

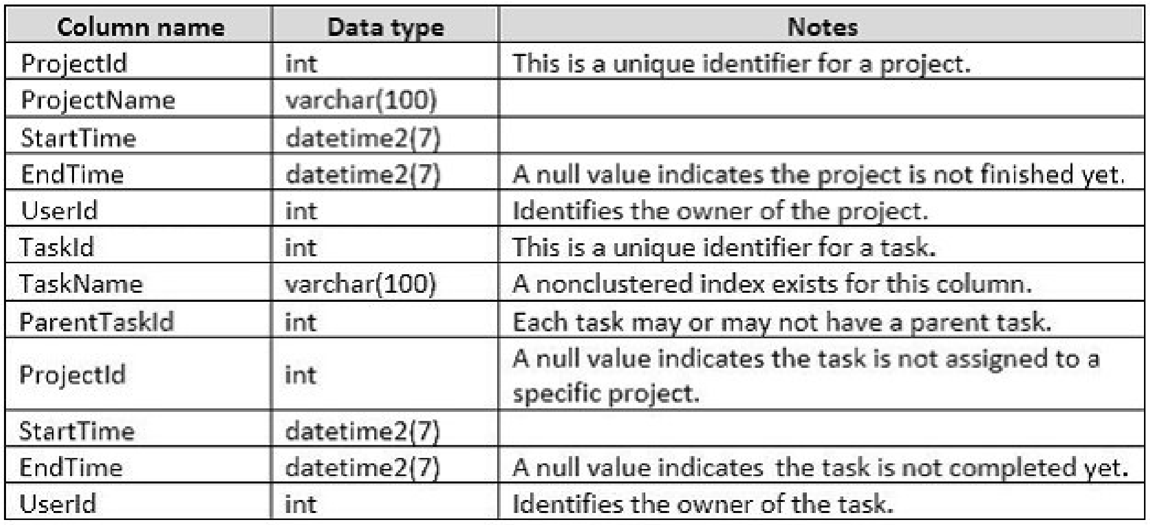
```
SELECT R.TaskId AS ParentTaskId,
T.TaskId, T.TaskName, R.TaskLevel+1
AS TaskLevel
FROM Task T INNER JOIN TaskWithLevel
R ON T.ParentTaskId = R.TaskId
```

```
With TaskWithLevel(ParentTaskId,
TaskId, TaskName, TaskLevel)
As (
```

```
)
SELECT * FROM TaskWithLevel
```

Explanation:
Box 1: SELECT CAST (NULL AS INT) AS ParentTaskID, etc.
This statement selects all tasks with task level 0. The ParentTaskID could be null so we should use CAST (NULL AS INT) AS ParentTaskID.
Box 2: UNION
We should use UNION and not UNION ALL as we do not went duplicate rows. UNION specifies that multiple result sets are to be combined and returned as a single result set.
Incorrect: Not UNION ALL: ALL incorporates all rows into the results. This includes duplicates. If not specified, duplicate rows are removed.
Box 3, Box 4, Box 5:
These statements select all tasks with task level >0.
References:
https://msdn.microsoft.com/en-us/library/ms180026.aspx

QUESTION 13
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.
Drag and Drop Question
You query a database that includes two tables: Project and Task. The Project table includes the following columns:

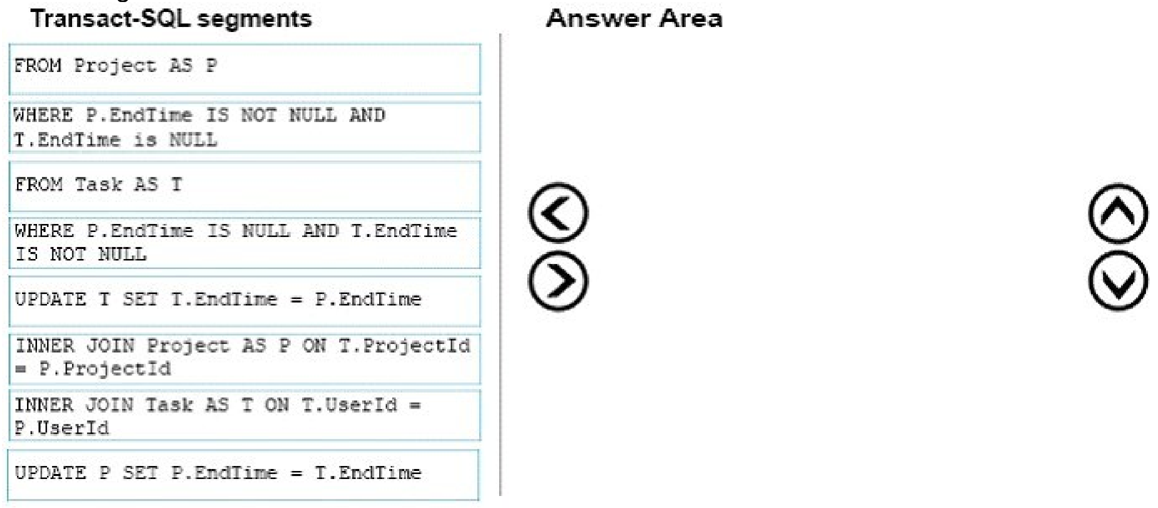


Column name	Data type	Notes
ProjectId	int	This is a unique identifier for a project.
ProjectName	varchar(100)	
StartTime	datetime2(7)	
EndTime	datetime2(7)	A null value indicates the project is not finished yet.
UserId	int	Identifies the owner of the project.
TaskId	int	This is a unique identifier for a task.
TaskName	varchar(100)	A nonclustered index exists for this column.
ParentTaskId	int	Each task may or may not have a parent task.
ProjectId	int	A null value indicates the task is not assigned to a specific project.
StartTime	datetime2(7)	
EndTime	datetime2(7)	A null value indicates the task is not completed yet.
UserId	int	Identifies the owner of the task.

When running an operation, you updated a column named EndTime for several records in the Project table, but updates to the corresponding task records in the Task table failed. You need to synchronize the value of the End Time column in the Task table with the value of the EndTime column in the project table. The solution must meet the following requirements:

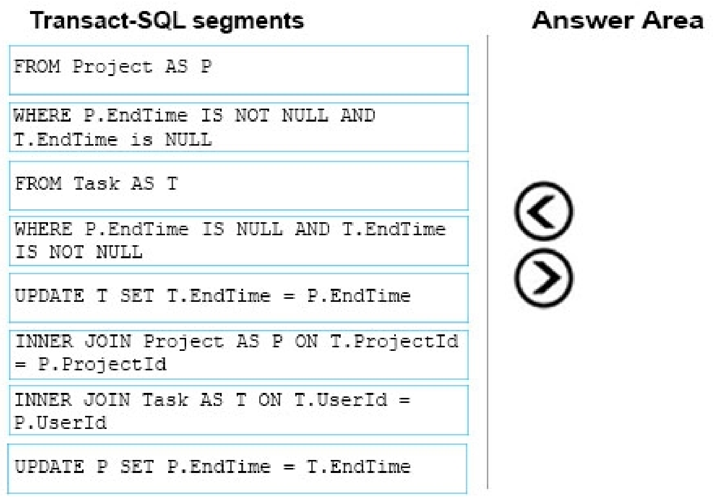
```
- If the End Time column has a value, make no changes to the record.
- If the value of the EndTime column is null and the corresponding project
record is marked as completed, update the record with the project finish
time.
```

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.



```
FROM Project AS P
```
```
WHERE P.EndTime IS NOT NULL AND
T.EndTime is NULL
```
```
FROM Task AS T
```
```
WHERE P.EndTime IS NULL AND T.EndTime
IS NOT NULL
```
```
UPDATE T SET T.EndTime = P.EndTime
```
```
INNER JOIN Project AS P ON T.ProjectId
= P.ProjectId
```
```
INNER JOIN Task AS T ON T.UserId =
P.UserId
```
```
UPDATE P SET P.EndTime = T.EndTime
```

Answer:



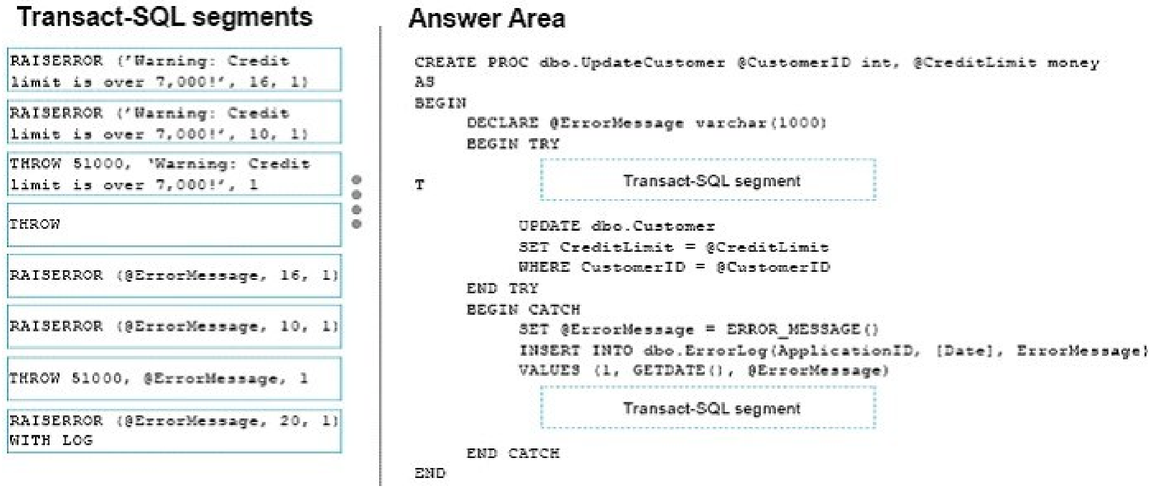
Transact-SQL segments

```
FROM Project AS P
```
```
WHERE P.EndTime IS NOT NULL AND
T.EndTime is NULL
```
```
FROM Task AS T
```
```
WHERE P.EndTime IS NULL AND T.EndTime
IS NOT NULL
```
```
UPDATE T SET T.EndTime = P.EndTime
```
```
INNER JOIN Project AS P ON T.ProjectId
= P.ProjectId
```
```
INNER JOIN Task AS T ON T.UserId =
P.UserId
```
```
UPDATE P SET P.EndTime = T.EndTime
```

Answer Area

Explanation:
Box 1: UPDATE T SET T.EndTime = P.EndTime
We are updating the EndTime column in the Task table.
Box 2: FROM Task AS T
Where are updating the task table.
Box 3: INNER JOIN Project AS P on T.ProjectID = P.ProjectID We join with the Project table (on the ProjectID columnID column).
Box 4: WHERE P.EndTime is NOT NULL AND T.EndTime is NULL We select the columns in the Task Table where the EndTime column in the Project table has a value (NOT NULL), but where it is NULL in the Task Table.
References: https://msdn.microsoft.com/en-us/library/ms177523.aspx

QUESTION 14
Drag and Drop Question
You need to create a stored procedure that meets the following requirements:
```
- Produces a warning if the credit limit parameter is greater than 7,000
- Propagates all unexpected errors to the calling process
```
How should you complete the Transact-SQL statement? To answer, drag the appropriate Transact-SQP 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.



Transact-SQL segments

```
RAISERROR ('Warning: Credit
limit is over 7,000!', 16, 1)
```
```
RAISERROR ('Warning: Credit
limit is over 7,000!', 10, 1)
```
```
THROW 51000, 'Warning: Credit
limit is over 7,000!', 1
```
```
THROW
```
```
RAISERROR (@ErrorMessage, 16, 1)
```
```
RAISERROR (@ErrorMessage, 10, 1)
```
```
THROW 51000, @ErrorMessage, 1
```
```
RAISERROR (@ErrorMessage, 20, 1)
WITH LOG
```

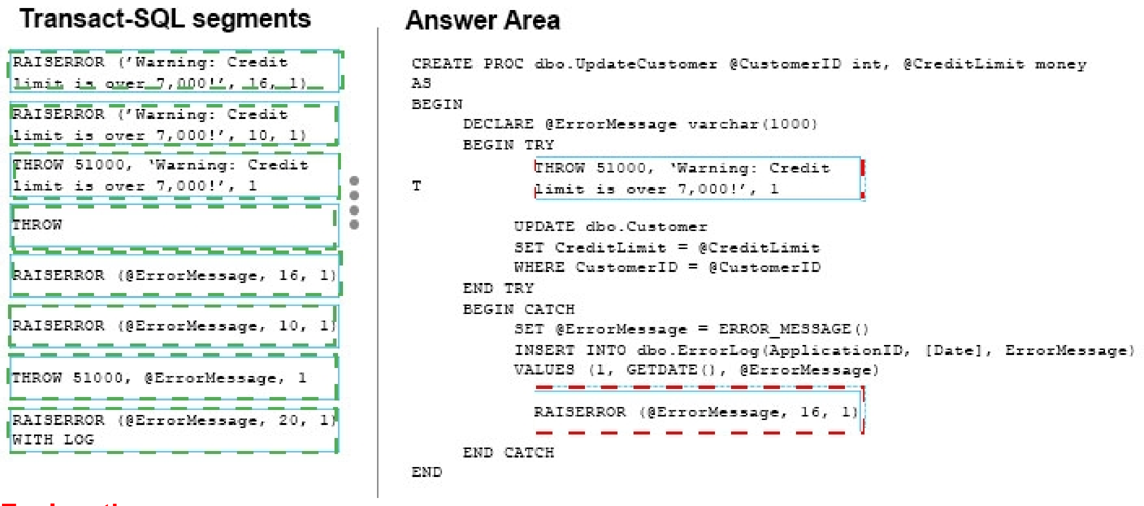
Answer Area

```
CREATE PROC dbo.UpdateCustomer @CustomerID int, @CreditLimit money
AS
BEGIN
     DECLARE @ErrorMessage varchar(1000)
     BEGIN TRY
T                  Transact-SQL segment

          UPDATE dbo.Customer
          SET CreditLimit = @CreditLimit
          WHERE CustomerID = @CustomerID
     END TRY
     BEGIN CATCH
          SET @ErrorMessage = ERROR_MESSAGE()
          INSERT INTO dbo.ErrorLog(ApplicationID, [Date], ErrorMessage)
          VALUES (1, GETDATE(), @ErrorMessage)
                   Transact-SQL segment

     END CATCH
END
```

Answer:



```
RAISERROR ('Warning: Credit
limit is over 7,000!', 16, 1)

RAISERROR ('Warning: Credit
limit is over 7,000!', 10, 1)

THROW 51000, 'Warning: Credit
limit is over 7,000!', 1

THROW

RAISERROR (@ErrorMessage, 16, 1)

RAISERROR (@ErrorMessage, 10, 1)

THROW 51000, @ErrorMessage, 1

RAISERROR (@ErrorMessage, 20, 1)
WITH LOG
```

```
CREATE PROC dbo.UpdateCustomer @CustomerID int, @CreditLimit money
AS
BEGIN
     DECLARE @ErrorMessage varchar(1000)
     BEGIN TRY
               THROW 51000, 'Warning: Credit
T              limit is over 7,000!', 1

          UPDATE dbo.Customer
          SET CreditLimit = @CreditLimit
          WHERE CustomerID = @CustomerID
     END TRY
     BEGIN CATCH
          SET @ErrorMessage = ERROR_MESSAGE()
          INSERT INTO dbo.ErrorLog(ApplicationID, [Date], ErrorMessage)
          VALUES (1, GETDATE(), @ErrorMessage)

            RAISERROR (@ErrorMessage, 16, 1)

     END CATCH
END
```

Explanation:
Box 1: THROW 51000, 'Warning: Credit limit is over 7,000!",1
THROW raises an exception and transfers execution to a CATCH block of a TRY...CATCH construct in SQL Server.
THROW syntax:
THROW [{ error_number | @local_variable },
{ message | @local_variable },
{ state | @local_variable }]
[;]
Box 2: RAISERROR (@ErrorMessage, 16,1)
RAISERROR generates an error message and initiates error processing for the session. RAISERROR can either reference a user-defined message stored in the sys.messages catalog view or build a message dynamically. The message is returned as a server error message to the calling application or to an associated CATCH block of a TRY...CATCH construct. New applications should use THROW instead.
Severity levels from 0 through 18 can be specified by any user. Severity levels from 19 through 25 can only be specified by members of the sysadmin fixed server role or users with ALTER TRACE permissions. For severity levels from 19 through 25, the WITH LOG option is required.
On Severity level 16. Using THROW to raise an exception The following example shows how to use the THROW statement to raise an exception.
Transact-SQL
THROW 51000, 'The record does not exist.', 1;
Here is the result set.
Msg 51000, Level 16, State 1, Line 1
The record does not exist.
Note: RAISERROR syntax:
RAISERROR ({ msg_id | msg_str | @local_variable } { ,severity ,state }
[,argument [,...n]])
[WITH option [,...n]]
Note: The ERROR_MESSAGE function returns the message text of the error that caused the CATCH block of a TRY...CATCH construct to be run.
References:
https://msdn.microsoft.com/en-us/library/ms178592.aspx
https://msdn.microsoft.com/en-us/library/ms190358.aspx
https://msdn.microsoft.com/en-us/library/ee677615.aspx

QUESTION 15

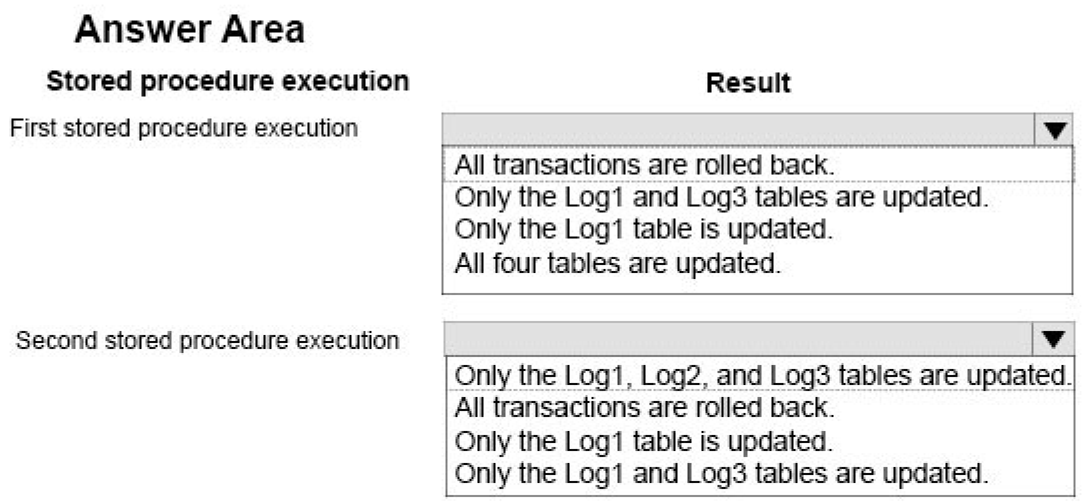
Hotspot Question
You have the following stored procedure:

```
CREATE PROC dbo.UpdateLogs @Code char(5), @ApplicationId int, @Info varchar(1000)
AS
BEGIN
     BEGIN TRY
          BEGIN TRAN
               INSERT INTO dbo.Log1 VALUES (@Code, @ApplicationId, @Info)
               IF @Code = 'C2323 AND @ApplicationId = 1
                    RAISERROR('C2323 code from HR application!', 16, 1)
               ELSE
                    INSERT INTO dbo.Log2 VALUES (@Code, @ApplicationId, @Info)
                    INSERT INTO dbo.Log3 VALUES (@Code, @ApplicationId, @Info)
                    BEGIN TRAN
                         IF @Code = 'C2323'
                              ROLLBACK TRAN
                         ELSE
                              INSERT INTO dbo.Log4 VALUES (@Code, @ApplicationId, @Info)
                              IF @@TRANCOUNT > 0
                                   COMMIT TRAN
     END TRY
     BEGIN CATCH
          IF XACT_STATE() != 0
               ROLLBACK TRAN
     END CATCH
END
```

You run the following Transact-SQL statements:

```
EXEC dbo.UpdateLogs 'C2323', 1, 'Employee records are updated.'
EXEC dbo.UpdateLogs 'C2323', 10, 'Sales process started.'
```

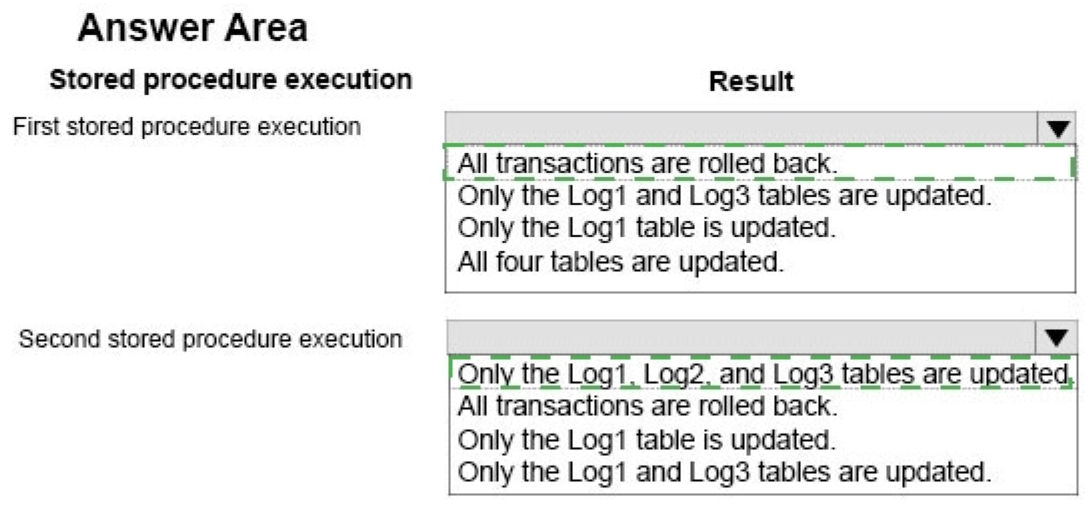
What is the result of each Transact-SQL statement? To answer, select the appropriate options in the answer area.



Answer Area

Stored procedure execution	Result
First stored procedure execution	All transactions are rolled back. Only the Log1 and Log3 tables are updated. Only the Log1 table is updated. All four tables are updated.
Second stored procedure execution	Only the Log1, Log2, and Log3 tables are updated. All transactions are rolled back. Only the Log1 table is updated. Only the Log1 and Log3 tables are updated.

Answer:

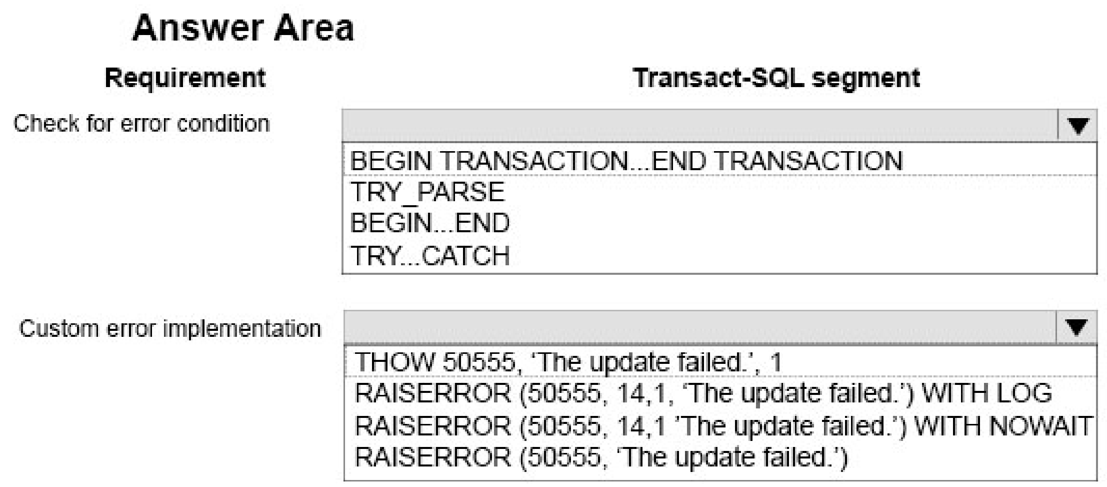


Answer Area

Stored procedure execution	Result
First stored procedure execution	[✓ All transactions are rolled back.] Only the Log1 and Log3 tables are updated. Only the Log1 table is updated. All four tables are updated.
Second stored procedure execution	[✓ Only the Log1, Log2, and Log3 tables are updated.] All transactions are rolled back. Only the Log1 table is updated. Only the Log1 and Log3 tables are updated.

Explanation:
Box 1: All transactions are rolled back.
The first IF-statement, IF @CODE = 'C2323' AND @ApplicationID = 1, will be true, an error will be raised, the error will be caught in the CATCH block, and the only transaction that has been started will be rolled back.
Box 2: Only Log1, Log2, and Log3 tables are updated. The second IF-statement, IF @Code = 'C2323', will be true, so the second transaction will be rolled back, but log1, log2, and log3 was updated before the second transaction.

QUESTION 16
Hotspot Question
You need to develop a Transact-SQL statement that meets the following requirements:
```
- The statement must return a custom error when there are problems updating a table.
- The error number must be value 50555.
- The error severity level must be 14.
- A Microsoft SQL Server alert must be triggered when the error condition occurs.
```
Which Transact-SQL segment should you use for each requirement? To answer, select the appropriate Transact-SQL segments in the answer area.



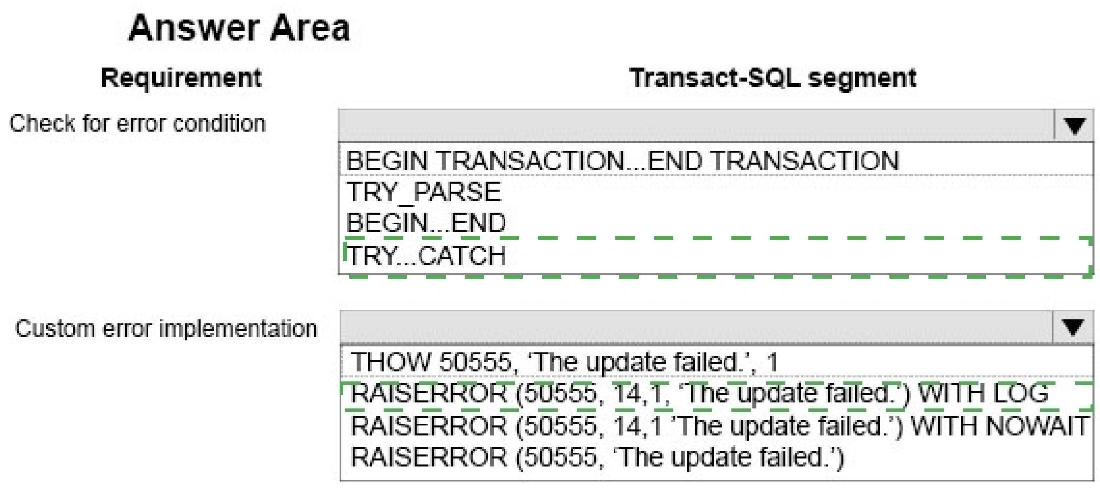
Answer Area

Requirement	Transact-SQL segment

Check for error condition	BEGIN TRANSACTION...END TRANSACTION
	TRY_PARSE
	BEGIN...END
	TRY...CATCH

Custom error implementation	THOW 50555, 'The update failed.', 1 RAISERROR (50555, 14,1, 'The update failed.') WITH LOG RAISERROR (50555, 14,1 'The update failed.') WITH NOWAIT RAISERROR (50555, 'The update failed.')

Answer:



Answer Area

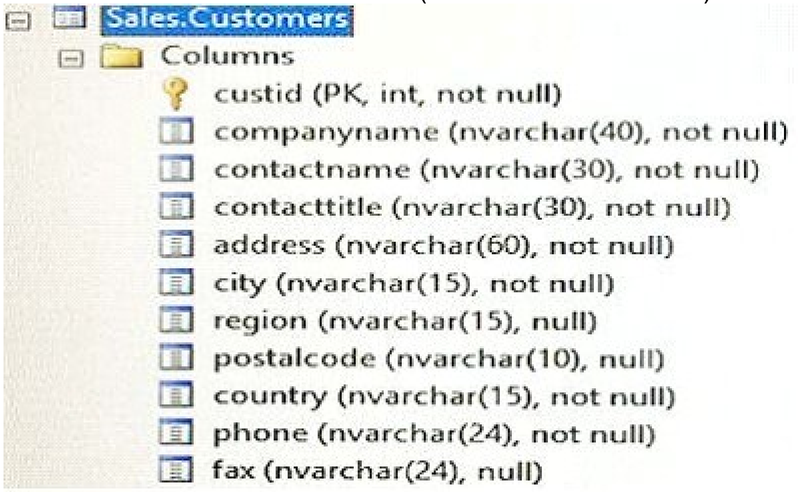
Requirement	Transact-SQL segment

Check for error condition	BEGIN TRANSACTION...END TRANSACTION
	TRY_PARSE
	BEGIN...END
	[✓ TRY...CATCH]

Custom error implementation	THOW 50555, 'The update failed.', 1 [✓ RAISERROR (50555, 14,1, 'The update failed.') WITH LOG] RAISERROR (50555, 14,1 'The update failed.') WITH NOWAIT RAISERROR (50555, 'The update failed.')

Explanation:
Box 1: TRY...CATCH
The TRY...CATCH Transact-SQL construct implements error handling for Transact-SQL that is similar to the exception handling in the Microsoft Visual C# and Microsoft Visual C++ languages. A group of Transact-SQL statements can be enclosed in a TRY block. If an error occurs in the TRY block, control is passed to another group of statements that is enclosed in a CATCH block.
Box 2: RAISERROR(50555, 14, 1 'The update failed.") WITH LOG We must use RAISERROR to be able to specify the required severity level of 14, and we should also use the LOG option, which Logs the error in the error log and the application log for the instance of the Microsoft SQL Server Database Engine, as this enable a MS MS SQL SERVER alert to be triggered.
Note: RAISERROR generates an error message and initiates error processing for the session. RAISERROR can either reference a user-defined message stored in the sys.messages catalog view or build a message dynamically. The message is returned as a server error message to the calling application or to an associated CATCH block of a TRY...CATCH construct.

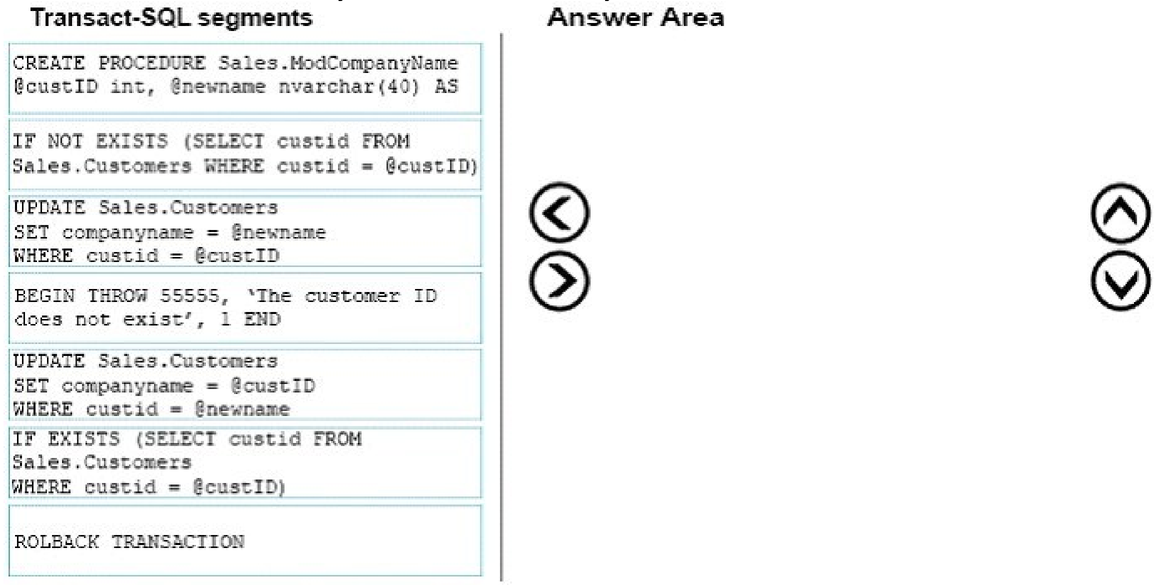
QUESTION 17
Drag and Drop Question
You need to create a stored procedure to update a table named Sales.Customers. The structure of the table is shown in the exhibit. (Click the exhibit button.)



```
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.



Transact-SQL segments

Answer Area

```
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
```

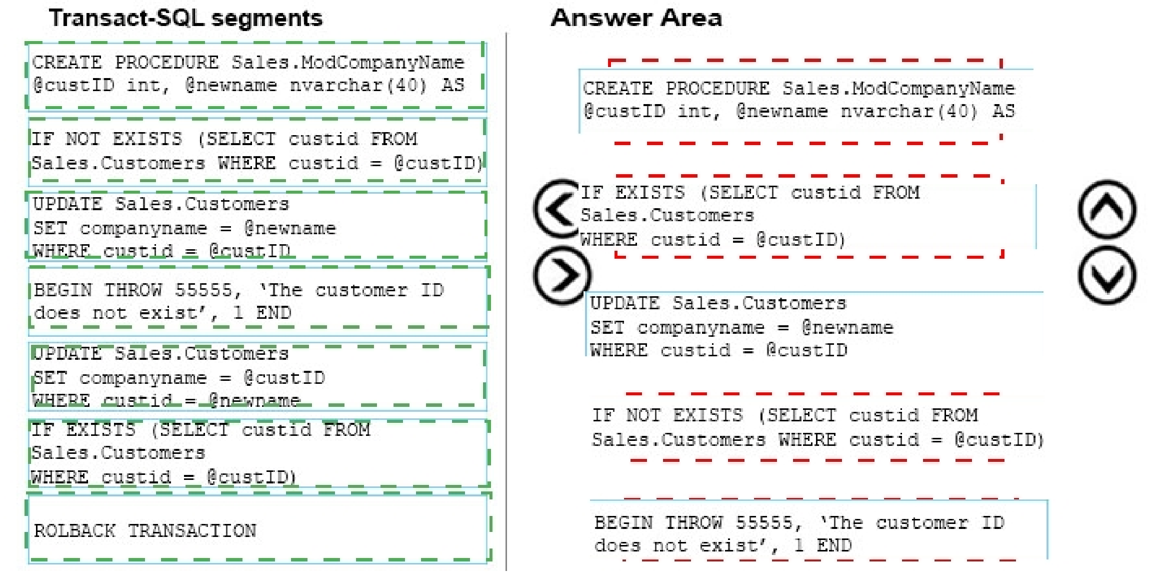
```
BEGIN THROW 55555, 'The customer ID
does not exist', 1 END
```

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

```
IF EXISTS (SELECT custid FROM
Sales.Customers
WHERE custid = @custID)
```

```
ROLBACK TRANSACTION
```

Answer:



Transact-SQL segments

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 18

You need to create an indexed view that requires logic statements to manipulate the data that the view displays. Which two database objects should you use? Each correct answer presents a complete solution.

A. a user-defined table-valued function
B. a CRL function

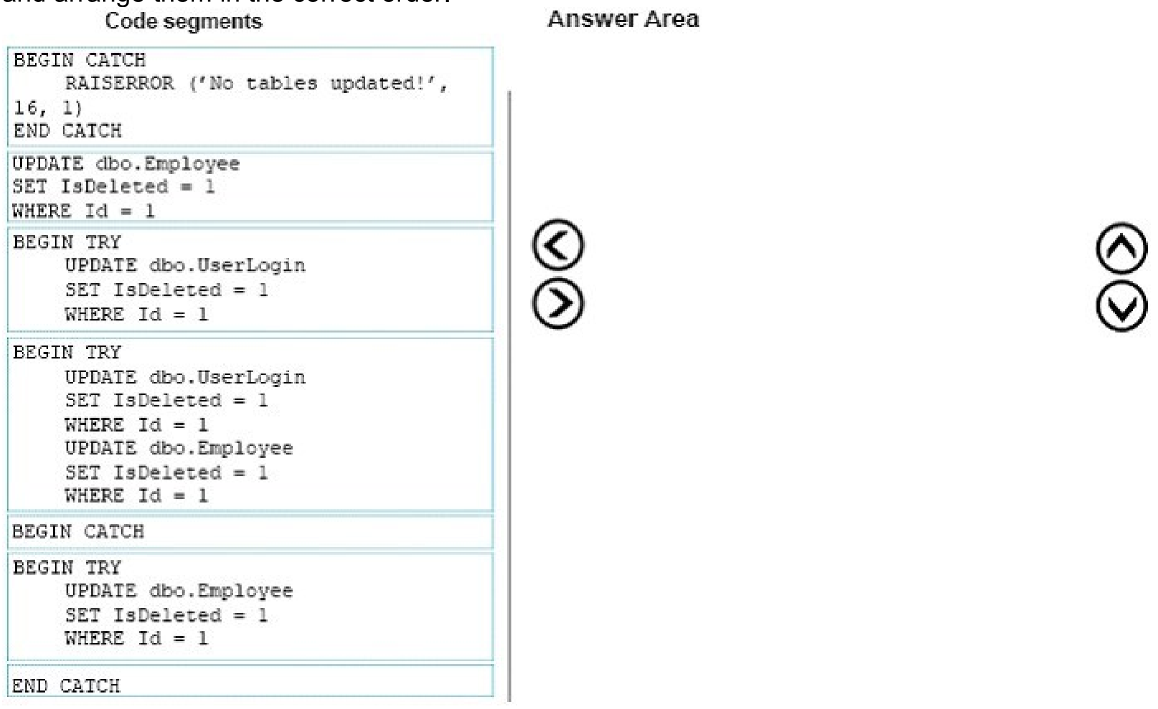
C. a stored procedure
D. a user-defined scalar function

Answer: AC

QUESTION 19
Drag and Drop Question
You have two tables named UserLogin and Employee respectively. You need to create a Transact-SQL script that meets the following requirements:

```
- The script must update the value of the IsDeleted column for the
UserLogin table to 1 if the value of the Id column for the User Login
table is equal to 1.
- The script must update the value of the IsDeleted column of the Employee
table to 1 if the value of the Id column is equal to 1 for the Employee
table when an update to the User Login table throws an error.
- The error message "No tables updated!" must be produced when an update
to the Employee table throws an error.
```

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 answerarea and arrange them in the correct order.



```
BEGIN CATCH
    RAISERROR ('No tables updated!',
16, 1)
END CATCH
```

```
UPDATE dbo.Employee
SET IsDeleted = 1
WHERE Id = 1
```

```
BEGIN TRY
    UPDATE dbo.UserLogin
    SET IsDeleted = 1
    WHERE Id = 1
```

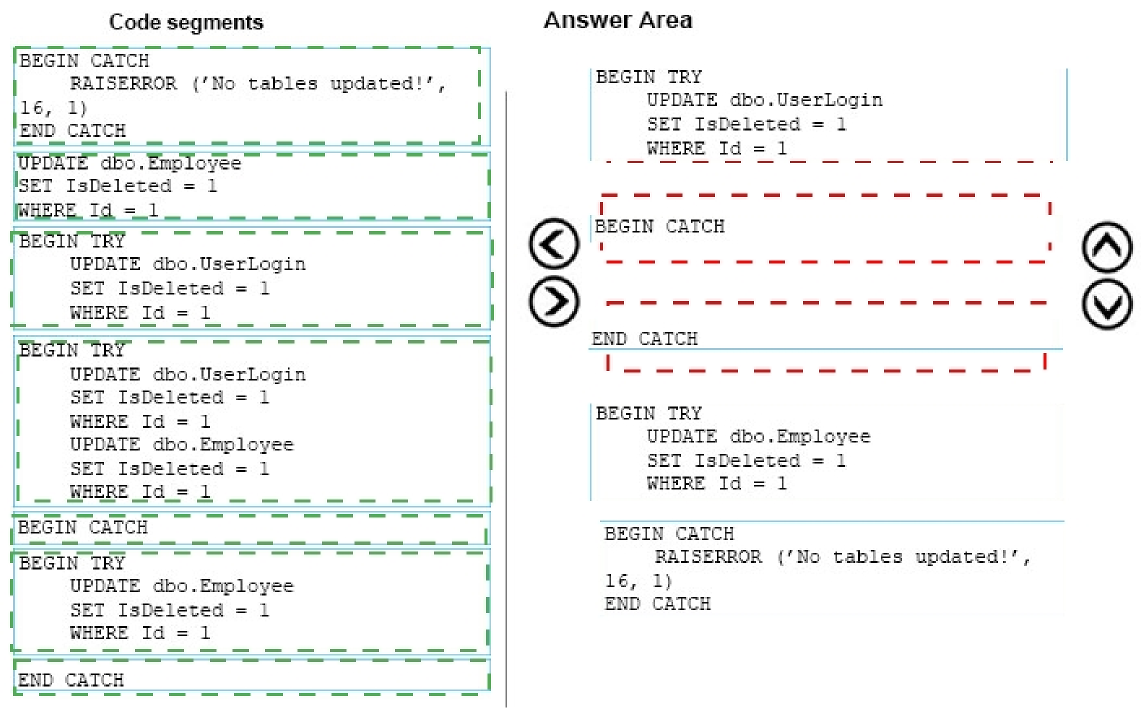
```
BEGIN TRY
    UPDATE dbo.UserLogin
    SET IsDeleted = 1
    WHERE Id = 1
    UPDATE dbo.Employee
    SET IsDeleted = 1
    WHERE Id = 1
```

```
BEGIN CATCH
```

```
BEGIN TRY
    UPDATE dbo.Employee
    SET IsDeleted = 1
    WHERE Id = 1
```

```
END CATCH
```

Answer:



Code segments

```
BEGIN CATCH
    RAISERROR ('No tables updated!',
16, 1)
END CATCH
```

```
UPDATE dbo.Employee
SET IsDeleted = 1
WHERE Id = 1
```

```
BEGIN TRY
    UPDATE dbo.UserLogin
    SET IsDeleted = 1
    WHERE Id = 1
```

```
BEGIN TRY
    UPDATE dbo.UserLogin
    SET IsDeleted = 1
    WHERE Id = 1
    UPDATE dbo.Employee
    SET IsDeleted = 1
    WHERE Id = 1
```

```
BEGIN CATCH
```

```
BEGIN TRY
    UPDATE dbo.Employee
    SET IsDeleted = 1
    WHERE Id = 1
```

```
END CATCH
```

Answer Area

```
BEGIN TRY
    UPDATE dbo.UserLogin
    SET IsDeleted = 1
    WHERE Id = 1

BEGIN CATCH

END CATCH

BEGIN TRY
    UPDATE dbo.Employee
    SET IsDeleted = 1
    WHERE Id = 1

BEGIN CATCH
    RAISERROR ('No tables updated!',
16, 1)
END CATCH
```

Explanation:
A TRY block must be immediately followed by an associated CATCH block. Including any other statements between the END TRY and BEGIN CATCH statements generates a syntax error.
References: https://msdn.microsoft.com/en-us/library/ms175976.aspx

QUESTION 20
You work for an organization that monitors seismic activity around volcanos. You have a table named GroundSensors. The table stored data collected from seismic sensors. It includes the columns describes in the following table:

Name	Data Type	Notes
SensorID	int	primary key
Location	geography	do not allow null values
Tremor	int	do not allow null values
NormalizedReading	float	allow null values

The database also contains a scalar value function named NearestMountain that returns the name of the mountain that is nearest to the sensor. You need to create a query that shows the average of the normalized readings from the sensors for each mountain. The query must meet the following requirements:

```
- Include the average normalized readings and nearest mountain name.
- Exclude sensors for which no normalized reading exists.
- Exclude those sensors with value of zero for tremor.
```

Construct the query using the following guidelines:

```
- Use one part names to reference tables, columns and functions.
- Do not use parentheses unless required.
- Do not use aliases for column names and table names.
- Do not surround object names with square brackets.
```

Keywords

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

```
select
```

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

Answer: `GROUP BY`
Explanation:
GROUP BY is a SELECT statement clause that divides the query result into groups of rows, usually for the purpose of performing one or more aggregations on each group. The SELECT statement returns one row per group.
References: https://msdn.microsoft.com/en-us/library/ms177673.aspx