



BrainDumps  
Collection

# Microsoft

## 70-761 Exam

Microsoft Querying Data with Transact-SQL Exam

Thank you for Downloading 70-761 exam PDF Demo

You can also try our 70-761 practice exam software

**Download Free Demo**

<https://www.braindumpscollection.com/70-761.html>

DEMO  
VERSION

(LIMITED CONTENT)

Questions  
& Answers

# Version: 17.1

---

## Question: 1

---

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 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.

Column name	Data type	Notes
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 plan to run the following query to update tasks that are not yet started:

```
UPDATE Task SET StartTime = GETDATE() WHERE StartTime IS NULL
```

You need to return the total count of tasks that are impacted by this UPDATE operation, but are not associated with a project.

What set of Transact-SQL statements should you run?

- ☐ A `DECLARE @startedTasks TABLE(ProjectId int)`  
`UPDATE Task SET StartTime = GETDATE() OUTPUT deleted.ProjectId INTO @startedTasks WHERE StartTime is NULL`  
`SELECT COUNT(*) FROM @startedTasks WHERE ProjectId IS NOT NULL`
- ☐ B `DECLARE @startedTasks TABLE(TaskId int, ProjectId int)`  
`UPDATE Task SET StartTime = GETDATE() OUTPUT deleted.TaskId, deleted.ProjectId INTO @startedTasks`  
`WHERE StartTime is NULL`  
`SELECT COUNT(*) FROM @startedTasks WHERE ProjectId IS NULL`
- ☐ C `DECLARE @startedTasks TABLE(TaskId int)`  
`UPDATE Task SET StartTime = GETDATE() OUTPUT inserted.TaskId, INTO @startedTasks WHERE StartTime is NULL`  
`SELECT COUNT(*) FROM @startedTasks WHERE TaskId IS NOT NULL`
- ☐ D `DECLARE @startedTasks TABLE(TaskId int)`  
`UPDATE Task SET StartTime = GETDATE() OUTPUT deleted.TaskId, INTO @startedTasks WHERE StartTime is NULL`  
`SELECT COUNT(*) FROM @startedTasks WHERE TaskId IS NOT NULL`

- A. Option A  
B. Option B  
C. Option C  
D. Option D

---

**Answer: B**

---

Explanation:

The WHERE clause of the third line should be WHERE ProjectID IS NULL, as we want to count the tasks that are not associated with a project.

---

## Question: 2

---

### HOTSPOT

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 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.

Column name	Data type	Notes
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,

ISNULL	( T.UserId, P.UserId, -1 P.UserId, T.UserId, -1 -1, P.UserId, T.UserId -1, T.UserId, P.UserId       ) AS OwnerUserId
COALESCE	
CHOOSE	

FROM Task T

	Project P ON T.ProjectId = P.ProjectId
INNER JOIN	
LEFT JOIN	
RIGHT JOIN	

---

**Answer:**

---

COALESCE  
 T.UserId, p.UserId, -1  
 LEFT 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: LEFT JOIN

The LEFT 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](http://www.w3schools.com/Sql/sql_join_right.asp)

---

### Question: 3

---

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 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.

Column name	Data type	Notes
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 EndTime 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 EndTime 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.

Transact-SQL segments	Answer Area
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:**

---

**Answer Area**

UPDATE T SET T.EndTime = P.EndTime

FROM Task AS T

INNER JOIN Project AS P ON T.ProjectId = P.ProjectId

WHERE P.EndTime IS NOT NULL AND T.EndTime is NULL

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: 4**

---

**DRAG DROP**

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-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.

**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
        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:**

---



## Answer Area

```

CREATE PROC dbo.UpdateCustomer @CustomerID int, @CreditLimit money
AS
BEGIN
    DECLARE @ErrorMessage varchar(1000)
    BEGIN TRY
        T
        THROW 51000, 'Warning: Credit
        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>

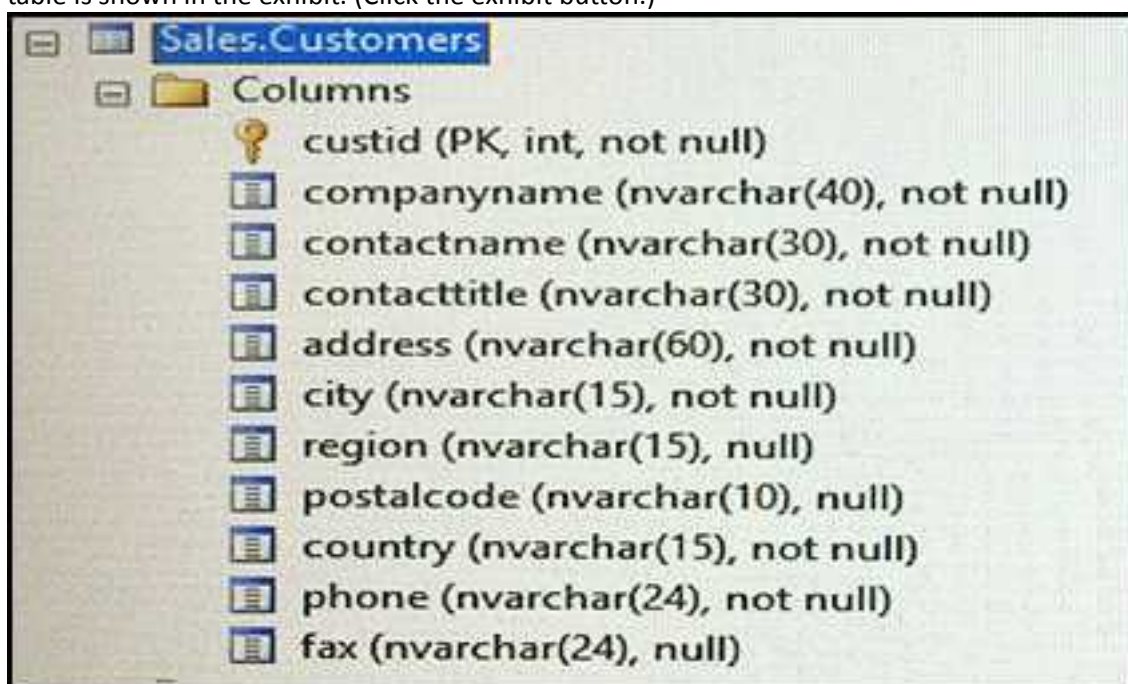
---












### Question: 5

---

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.

**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**

---

**Answer:**

---

**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: 6**

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: A,C**

---



---

### Question: 7

---

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

You need to create a query that meets the following requirements:

- For customers that are not on a credit hold, return the CustomerID and the latest recorded population for the delivery city that is associated with the customer.
- For customers that are on a credit hold, return the CustomerID and the latest recorded population for the postal city that is associated with the customer.

Which two Transact-SQL queries will achieve the goal? Each correct answer presents a complete solution.

- A**
- ```
SELECT CustomerID, LatestRecordedPopulation
FROM Sales.Customers
CROSS JOIN Application.Cities
WHERE (IsOnCreditHold = 0 AND DeliveryCityID = CityID)
OR (IsOnCreditHold = 1 AND PostalCityID = CityID)
```
- B**
- ```
SELECT CustomerID, LatestRecordedPopulation
FROM Sales.Customers
INNER JOIN Application.Cities AS A
ON A.CityID = IIF(IsOnCreditHold = 0, DeliveryCityID, PostalCityID)
```
- C**
- ```
SELECT CustomerID, ISNULL(A.LatestRecordedPopulation, B.LatestRecordedPopulation)
FROM Sales.Customers
INNER JOIN Application.Cities AS A ON A.CityID = DeliveryCityID
INNER JOIN Application.Cities AS B ON B.CityID = PostalCityID
WHERE IsOnCreditHold = 0
```
- D**
- ```
SELECT CustomerID, LatestRecordedPopulation,
IIF(IsOnCreditHold = 0, DeliveryCityID, PostalCityID) AS CityId
FROM Sales.Customers
INNER JOIN Application.Cities AS A ON A.CityID = CityId
```

- A. Option A
- B. Option B
- C. Option C
- D. Option D

---

**Answer: AB**

---

Explanation:

Using Cross Joins

A cross join that does not have a WHERE clause produces the Cartesian product of the tables involved in the join. The size of a Cartesian product result set is the number of rows in the first table multiplied by the number of rows in the second table.

However, if a WHERE clause is added, the cross join behaves as an inner join.

B: You can use the IIF in the ON-statement.

IIF returns one of two values, depending on whether the Boolean expression evaluates to true or false in SQL Server.

References:

[https://technet.microsoft.com/en-us/library/ms190690\(v=sql.105\).aspx](https://technet.microsoft.com/en-us/library/ms190690(v=sql.105).aspx)

<https://msdn.microsoft.com/en-us/library/hh213574.aspx>

## Question: 8

### 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.)

Column Name	Data Type	Allow Nulls
SalesSummaryKey	int	<input type="checkbox"/>
SalesYear	smallint	<input type="checkbox"/>
SalesQuarter	smallint	<input type="checkbox"/>
SalesMonth	smallint	<input type="checkbox"/>
SalesDate	date	<input type="checkbox"/>
ProductCode	char(12)	<input type="checkbox"/>
CustomerCode	char(6)	<input type="checkbox"/>
EmployeeCode	char(6)	<input type="checkbox"/>
RegionCode	char(2)	<input checked="" type="checkbox"/>
SalesAmount	money	<input type="checkbox"/>

Column Name	Data Type	Allow Nulls
EmployeeID	smallint	<input type="checkbox"/>
EmployeeCode	char(6)	<input type="checkbox"/>
FirstName	varchar(30)	<input checked="" type="checkbox"/>
MiddleName	varchar(30)	<input checked="" type="checkbox"/>
LastName	varchar(40)	<input type="checkbox"/>
Title	varchar(50)	<input type="checkbox"/>
ManagerID	smallint	<input checked="" type="checkbox"/>

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 word 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 columns 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 to supply data for the Sales Hierarchy report.

Which Transact-SQL keyword should you use for each operation? To answer, drag the appropriate Transact-SQL keywords to the correct operations. Each action may be used once, more than once or not at all. You may drag the split bar between panes or scroll to view content.



Transact-SQL keywords		Answer Area								
GROUP BY		<table border="1"> <thead> <tr> <th>Operation</th> <th>Transact-SQL keyword</th> </tr> </thead> <tbody> <tr> <td>Aggregation</td> <td>Transact-SQL keyword</td> </tr> <tr> <td>Grouping</td> <td>Transact-SQL keyword</td> </tr> <tr> <td>Filtering</td> <td>Transact-SQL keyword</td> </tr> </tbody> </table>	Operation	Transact-SQL keyword	Aggregation	Transact-SQL keyword	Grouping	Transact-SQL keyword	Filtering	Transact-SQL keyword
Operation	Transact-SQL keyword									
Aggregation	Transact-SQL keyword									
Grouping	Transact-SQL keyword									
Filtering	Transact-SQL keyword									
ROLLUP	⏪	⏩								
CHECKSUM_AGG	⏩	⏪								
SUM										
GROUPING_ID										
GROUPING										

---

**Answer:**

---

Operation	Transact-SQL keyword
Aggregation	SUM
Grouping	GROUP BY
Filtering	ROLLUP

Reference:

<https://docs.microsoft.com/en-us/sql/t-sql/functions/sum-transact-sql?view=sql-server-ver15>

<https://www.sqlservertutorial.net/sql-server-basics/sql-server-rollup/>

---

### Question: 9

---

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

a. 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

You discover an application bug that impacts customer data for records created on or after January 1, 2014. In order to fix the data impacted by the bug, application programmers require a report that contains customer data as it existed on December 31, 2013.

You need to provide the query for the report.

Which Transact-SQL statement should you use?

- A**
- ```
DECLARE @sdate DATETIME, @edate DATETIME
SET @sdate = DATEFROMPARTS (2013, 12, 31)
set @edate = DATEADD(d, 1, @sdate)
SELECT * FROM Sales.Customers FOR SYSTEM_TIME ALL
WHERE ValidFrom > @sdate AND ValidTo < @edate
```
- B**
- ```
DECLARE @sdate DATETIME, @edate DATETIME
SET @sdate = DATEFROMPARTS (2013, 12, 31)
set @edate = DATEADD(d, -1, @sdate)
SELECT * FROM Sales.Customers FOR SYSTEM_TIME BETWEEN @sdate AND @edate
```
- C**
- ```
DECLARE @date DATE
SET @date = DATEFROMPARTS (2013, 12, 31)
SELECT * FROM Sales.Customers FOR SYSTEM_TIME AS OF @date
```
- D**
- ```
DECLARE @date DATE
SET @date = DATEFROMPARTS (2013, 12, 31)
SELECT * FROM Sales.Customers WHERE @date BETWEEN ValidFrom AND ValidTo
```

- A. Option A  
B. Option B  
C. Option C  
D. Option D

---

**Answer: D**

---

Explanation:

The datetime datatype defines a date that is combined with a time of day with fractional seconds that is based on a 24-hour clock.

The DATEFROMPARTS function returns a date value for the specified year, month, and day.

### 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.

a. 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

You are creating a report to measure the impact of advertising efforts that were designed to attract new customers. The report must show the number of new customers per day for each customer category, but only if the number of new customers is greater than five.

You need to write the query to return data for the report.

How should you complete the Transact-SQL statement? 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.

**Transact-SQL segments**

CAST (Cust.AccountOpenedDate  
AS DATE)

DATEPART (day,  
Cust.AccountOpenedDate)

HAVING

WHERE

COUNT (Cust.CustomerId)

MAX (Cust.CustomerID)

RANK

GROUP BY

**Answer Area**

```
SELECT Count (Cust.CustomerId), CustCat.CustomerCategoryName,  
FROM Sales.Customers AS Cust  
INNER JOIN Sales.CustomerCategories AS CustCat  
ON Cust.CustomerCategoryID = CustCat.CustomerCategoryID
```

Transact-SQL segment

CustCat.CustomerCategoryName,

Transact-SQL segment

Transact-SQL segment

Transact-SQL segment

> 5

---

**Answer:**

---

**Answer Area**

```
SELECT Count (Cust.CustomerId), CustCat.CustomerCategoryName,  
FROM Sales.Customers AS Cust  
INNER JOIN Sales.CustomerCategories AS CustCat  
ON Cust.CustomerCategoryID = CustCat.CustomerCategoryID
```

GROUP BY

CustCat.CustomerCategoryName,

CAST (Cust.AccountOpenedDate  
AS DATE)

WHERE

COUNT (Cust.CustomerId)

> 5

**Thank You for trying 70-761 PDF Demo**

To try our 70-761 practice exam software visit link below

<https://www.braindumpscollection.com/70-761.html>

## Start Your 70-761 Preparation

Use Coupon "20OFF" for extra 20% discount on the purchase of Practice Test Software. Test your 70-761 preparation with actual exam questions.