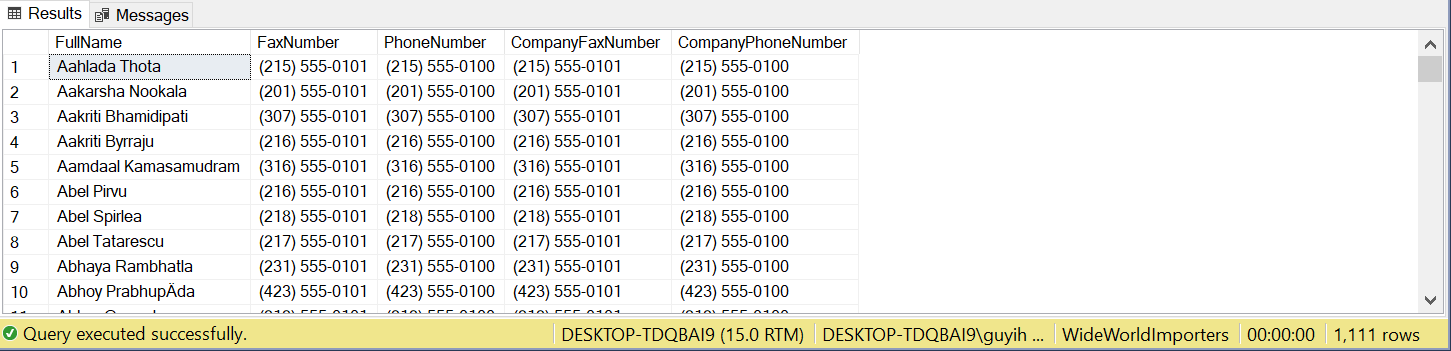
1.

SELECT DISTINCT p.FullName, p.FaxNumber, p.PhoneNumber, c.FaxNumber AS CompanyFaxNumber, c.PhoneNumber AS CompanyPhoneNumber

FROM Application.People p

LEFT JOIN Sales.Customers c ON p.PersonID = c.PrimaryContactPersonID

OR p.PersonID = c.AlternateContactPersonID



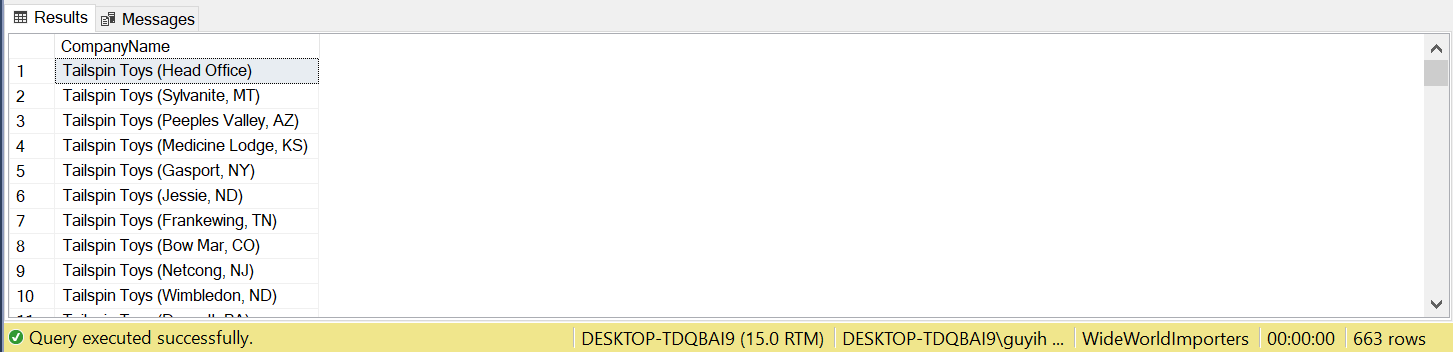
2.

SELECT CustomerName AS CompanyName

FROM Sales.Customers c

JOIN Application.People p ON c.PrimaryContactPersonID = p.PersonID

AND c.PhoneNumber = p.PhoneNumber



3.

SELECT DISTINCT CustomerID

FROM Sales.Orders

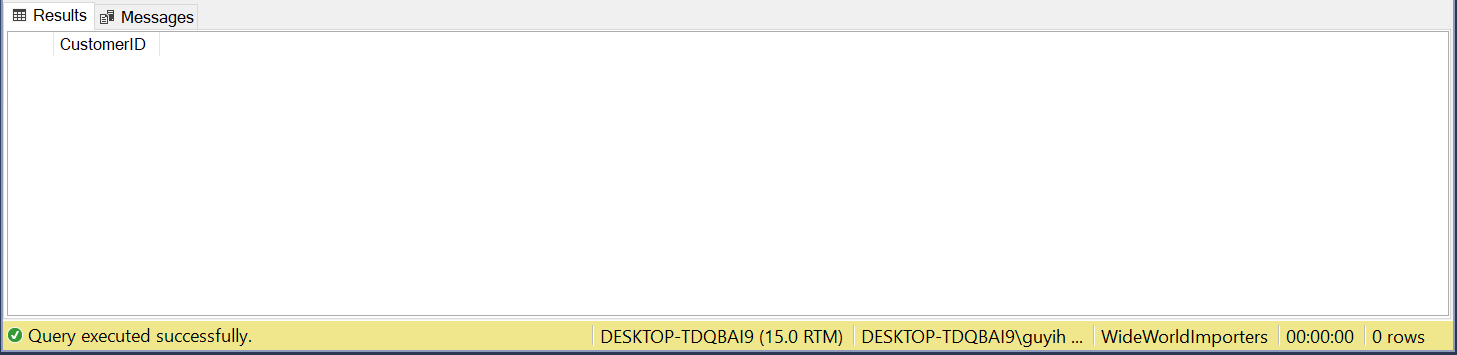
WHERE YEAR(OrderDate) < 2016

AND CustomerID NOT IN (

SELECT DISTINCT CustomerID

FROM Sales.Orders

WHERE YEAR(OrderDate) >= 2016)



4.

SELECT i.StockItemName, SUM(ol.OrderedOuters) AS TotalQuantity

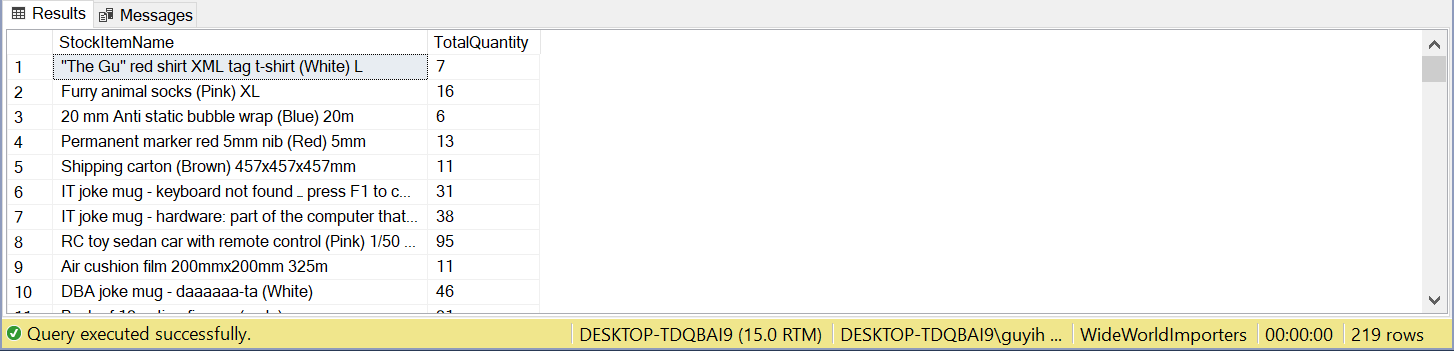
FROM Purchasing.PurchaseOrderLines ol

JOIN Purchasing.PurchaseOrders o ON ol.PurchaseOrderID = o.PurchaseOrderID

JOIN Warehouse.StockItems i ON i.StockItemID = ol.StockItemID

WHERE YEAR(o.OrderDate) = 2013

GROUP BY i.StockItemName

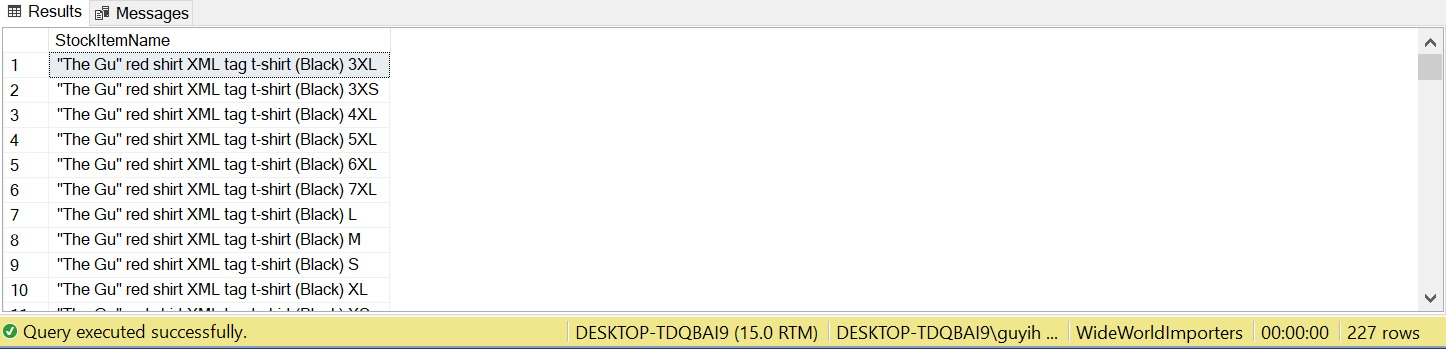


5.

SELECT StockItemName

FROM Warehouse.StockItems

WHERE LEN(StockItemName) > 10



6.

WITH cte AS(

SELECT ol.StockItemID, s.StateProvinceName

FROM Sales.Customers c

JOIN Sales.Orders o ON c.CustomerID = o.CustomerID AND YEAR(o.OrderDate) = 2014

JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID

JOIN Application.Cities ci ON c.DeliveryCityID = ci.CityID

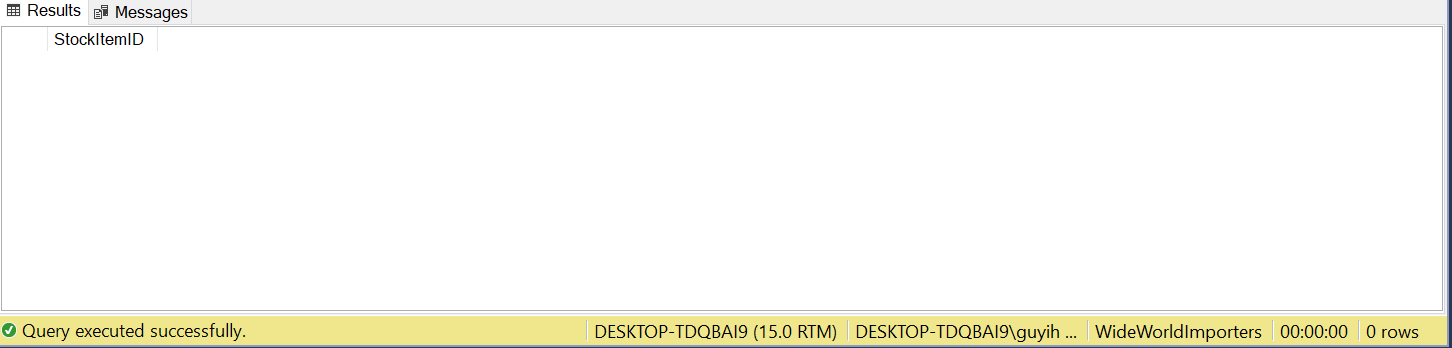
JOIN Application.StateProvinces s ON ci.StateProvinceID = s.StateProvinceID

)

SELECT DISTINCT StockItemID

FROM cte

WHERE StockItemID NOT IN (SELECT StockItemID FROM cte WHERE StateProvinceName IN ('Alabama', 'Georgia'))



7.

SELECT s.StateProvinceName, AVG(DATEDIFF(d, o.OrderDate, i.ConfirmedDeliveryTime)) AS AvgProDate

FROM Sales.Orders o

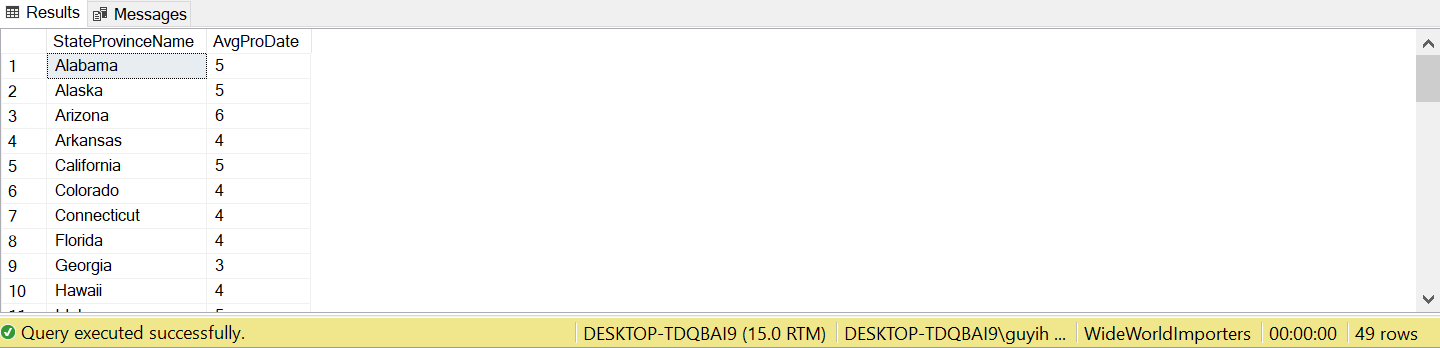
JOIN Sales.Invoices i ON o.OrderID = i.OrderID

JOIN Sales.Customers c ON c.CustomerID = o.CustomerID

JOIN Application.Cities ci ON ci.CityID = c.DeliveryCityID

JOIN Application.StateProvinces s ON ci.StateProvinceID = s.StateProvinceID

GROUP BY s.StateProvinceName



8.

SELECT s.StateProvinceName, MONTH(o.OrderDate) AS month,

AVG(DATEDIFF(d, o.OrderDate, i.ConfirmedDeliveryTime)) AS AvgProcessDay

FROM Sales.Orders o

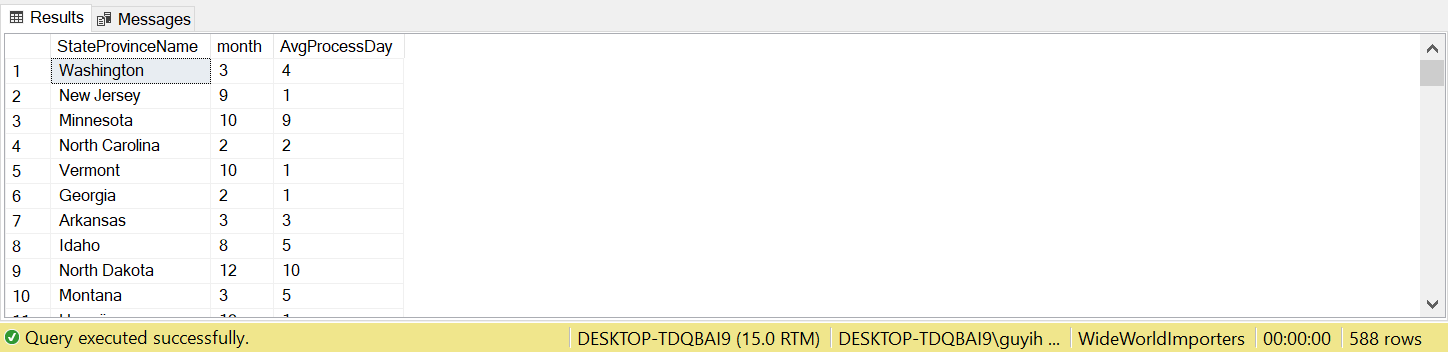
JOIN Sales.Invoices i ON o.OrderID = i.OrderID

JOIN Sales.Customers c ON c.CustomerID = o.CustomerID

JOIN Application.Cities ci ON ci.CityID = c.DeliveryCityID

JOIN Application.StateProvinces s ON ci.StateProvinceID = s.StateProvinceID

GROUP BY s.StateProvinceName, MONTH(o.OrderDate)



9.

WITH cte0 AS (

SELECT ol.StockItemID, SUM(ol.OrderedOuters) AS PurchaseQuantity

FROM Purchasing.PurchaseOrders o

JOIN Purchasing.PurchaseOrderLines ol ON o.PurchaseOrderID = ol.PurchaseOrderID

WHERE YEAR(o.OrderDate) = 2015

GROUP BY ol.StockItemID

),

cte1 AS (

SELECT ol.StockItemID, SUM(ol.Quantity) AS SaleQuantity

FROM Sales.Orders o

JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID

WHERE YEAR(o.OrderDate) = 2015

GROUP BY ol.StockItemID)

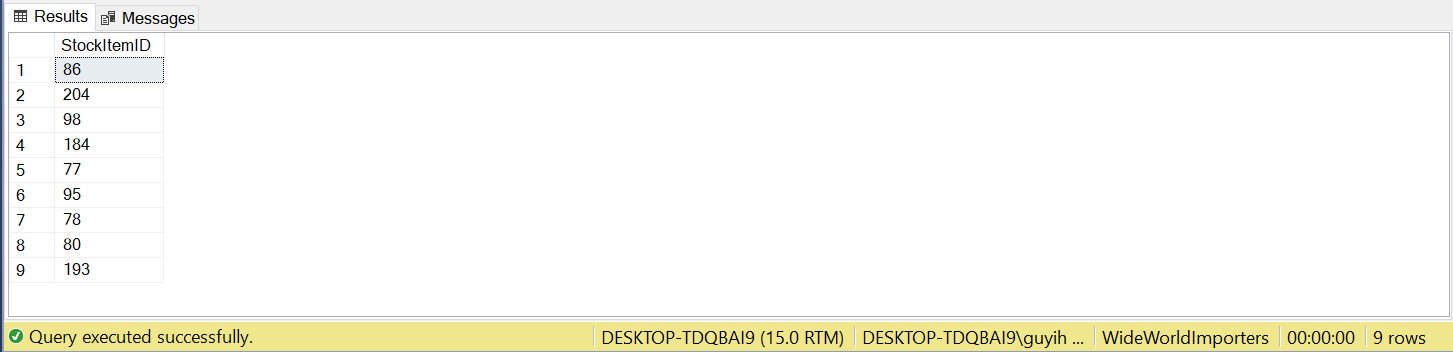
SELECT c0.StockItemID

FROM cte0 c0

LEFT JOIN cte1 c1 ON c0.StockItemID = c1.StockItemID

WHERE c1.SaleQuantity IS NULL

OR c0.PurchaseQuantity > c1.SaleQuantity



10.

WITH cte AS (

SELECT c.CustomerID

FROM Sales.Customers c

JOIN sales.Orders o ON c.CustomerID = o.CustomerID

JOIN sales.OrderLines ol ON ol.OrderID = o.OrderID

JOIN Warehouse.StockItemStockGroups s ON s.StockItemID = ol.StockItemID

JOIN Warehouse.StockGroups sg ON sg.StockGroupID = s.StockGroupID

WHERE YEAR(o.OrderDate) = 2016

AND sg.StockGroupName = 'Mugs'

GROUP BY c.CustomerID

HAVING COUNT(ol.Quantity) <= 10

)

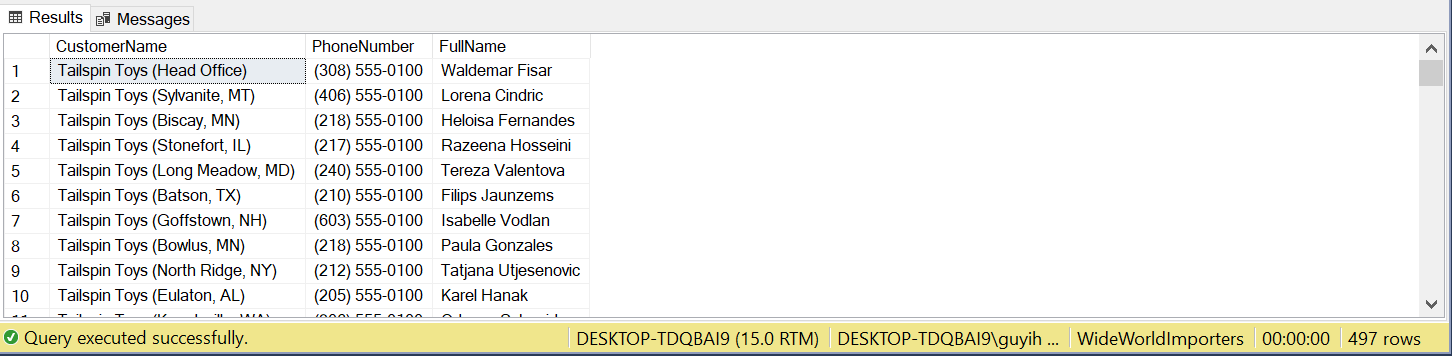
SELECT c.CustomerName, c.PhoneNumber, p.FullName

FROM sales.Customers c

JOIN Application.People p ON c.PrimaryContactPersonID = p.PersonID

WHERE c.CustomerID IN (SELECT CustomerID

FROM cte)

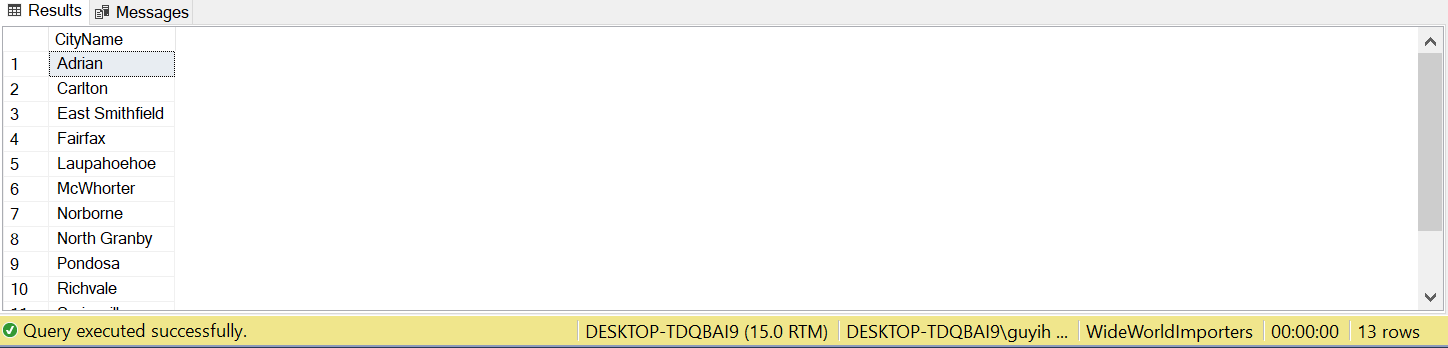


11.

SELECT CityName

FROM Application.Cities

WHERE ValidFrom > '2015-01-01'



12.

SELECT s.StockItemName, CONCAT(c.DeliveryAddressLine2, ' ', c.DeliveryAddressLine1) AS DeliveryAdress,

st.StateProvinceName, ci.CityName, co.CountryName, c.CustomerName, p.FullName, c.PhoneNumber, ol.Quantity

FROM Sales.OrderLines ol

JOIN Sales.Orders o ON ol.OrderID = o.OrderID

JOIN Sales.Customers

FOR SYSTEM\_TIME AS OF '2014-07-01' c ON c.CustomerID = o.CustomerID

JOIN Application.People

FOR SYSTEM\_TIME AS OF '2014-07-01' p ON c.PrimaryContactPersonID = p.PersonID

JOIN Warehouse.StockItems

FOR SYSTEM\_TIME AS OF '2014-07-01' s ON s.StockItemID = ol.StockItemID

JOIN Application.Cities

FOR SYSTEM\_TIME AS OF '2014-07-01' ci ON c.DeliveryCityID = ci.CityID

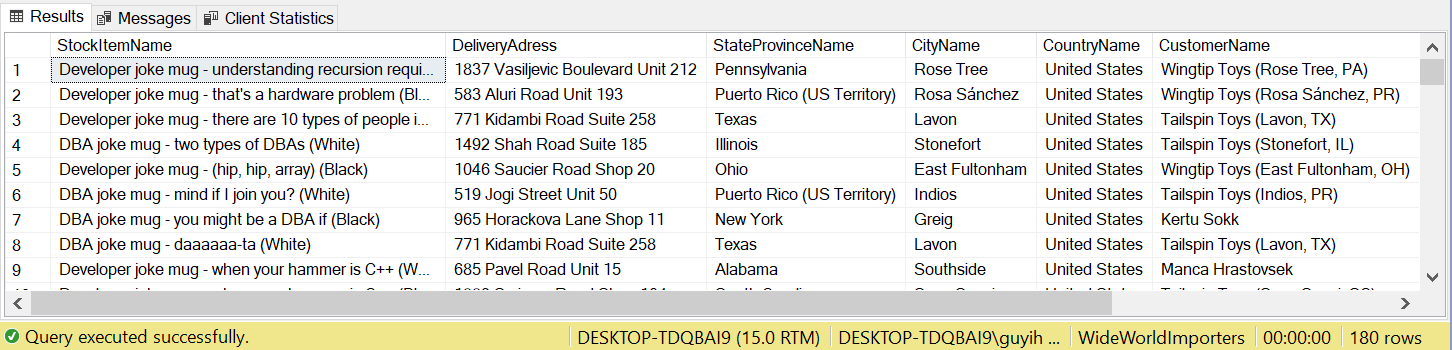
JOIN Application.StateProvinces

FOR SYSTEM\_TIME AS OF '2014-07-01' st ON ci.StateProvinceID = st.StateProvinceID

JOIN Application.Countries

FOR SYSTEM\_TIME AS OF '2014-07-01' co ON st.CountryID = co.CountryID

WHERE o.OrderDate = '2014-07-01'



13.

WITH cte0 AS (

SELECT s.StockGroupID, SUM(p.OrderedOuters) AS PurchaseQuantity

FROM Purchasing.PurchaseOrderLines p

JOIN Warehouse.StockItemStockGroups s ON p.StockItemID = s.StockItemID

GROUP BY s.StockGroupID

),

cte1 AS (

SELECT s.StockGroupID, SUM(o.Quantity) AS SaleQuantity

FROM Sales.OrderLines o

JOIN Warehouse.StockItemStockGroups s ON o.StockItemID = s.StockItemID

GROUP BY s.StockGroupID)

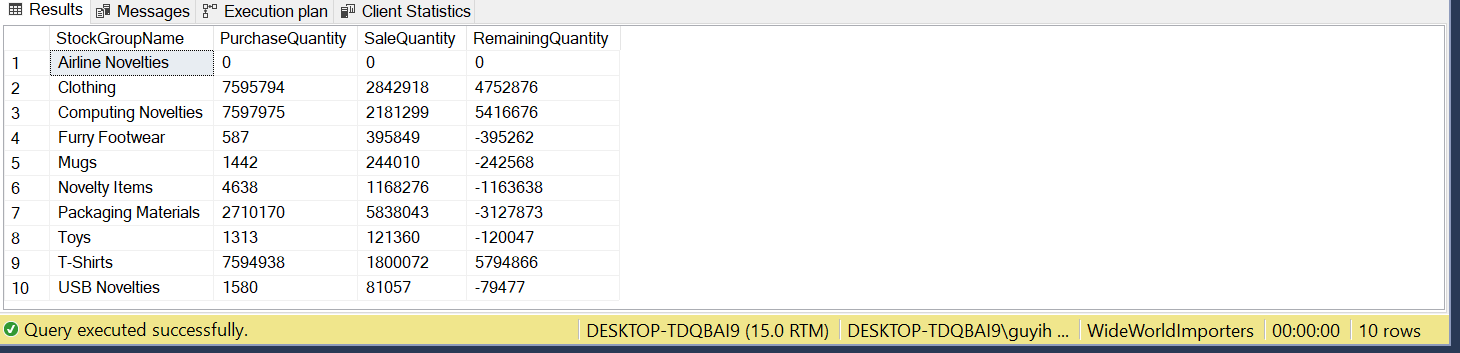
SELECT s.StockGroupName, ISNULL(c0.PurchaseQuantity, 0) AS PurchaseQuantity, ISNULL(c1.SaleQuantity, 0) AS SaleQuantity,

ISNULL(c0.PurchaseQuantity, 0) - ISNULL(c1.SaleQuantity, 0) AS RemainingQuantity

FROM Warehouse.StockGroups s

LEFT JOIN cte0 c0 ON s.StockGroupID = c0.StockGroupID

LEFT JOIN cte1 c1 ON s.StockGroupID = c1.StockGroupID



14.

WITH cte0 AS (

SELECT ol.StockItemID, c.DeliveryCityID, COUNT(\*) AS Delivery

FROM Sales.OrderLines ol

JOIN Sales.Orders o ON o.OrderID = ol.OrderID

JOIN sales.Customers c ON o.CustomerID = c.CustomerID

WHERE YEAR(o.OrderDate) = 2016

GROUP BY ol.StockItemID, c.DeliveryCityID),

cte1 AS(

SELECT StockItemID, DeliveryCityID

FROM (

SELECT StockItemID, DeliveryCityID,

DENSE\_RANK() OVER(PARTITION BY DeliveryCityId ORDER BY Delivery DESC) AS rnk

FROM cte0) a

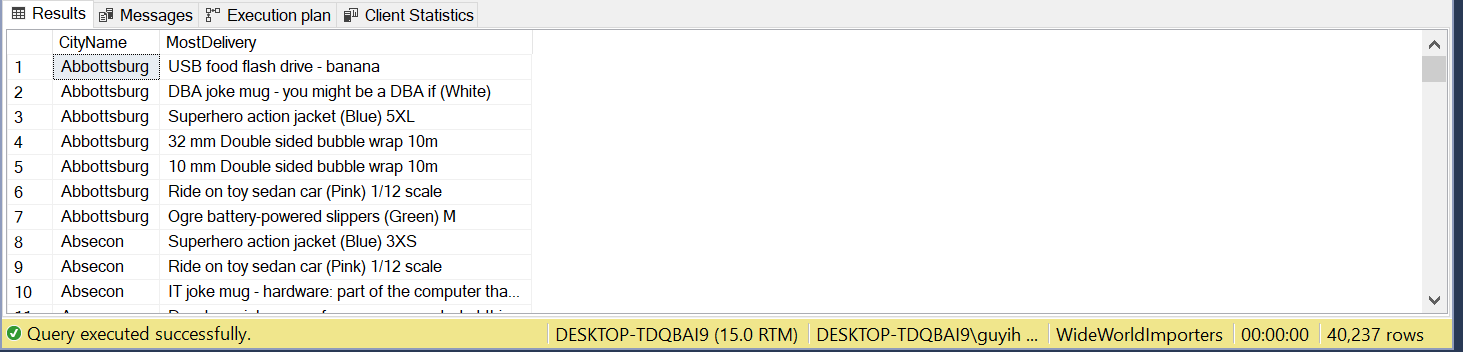
WHERE rnk = 1

)

SELECT c.CityName, ISNULL(s.StockItemName, 'No Sale') AS MostDelivery

FROM cte1 c1 JOIN Warehouse.StockItems s ON c1.StockItemID = s.StockItemID

RIGHT JOIN Application.Cities c ON c1.DeliveryCityID = c.CityID

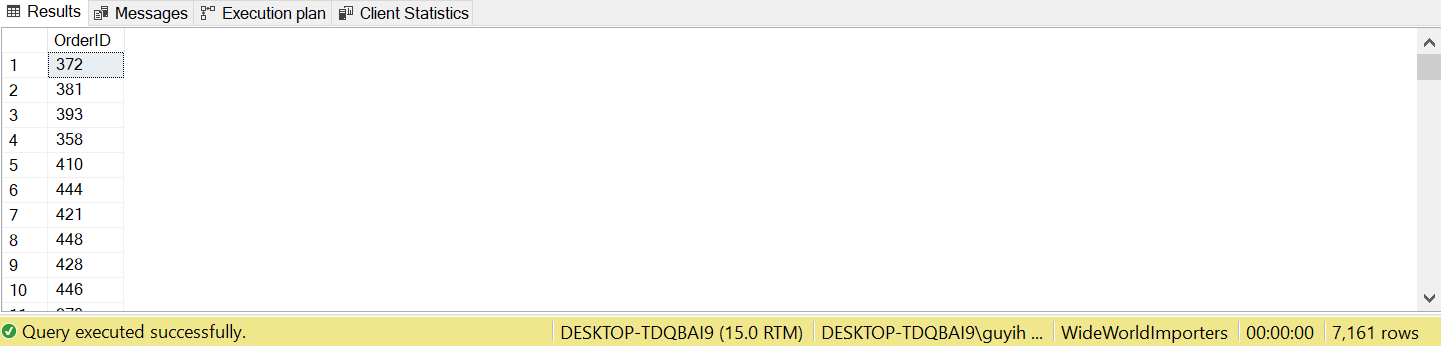


15.

SELECT OrderID

FROM Sales.Invoices

WHERE JSON\_VALUE(ReturnedDeliveryData, '$.Events[1].Comment') IS NOT NULL

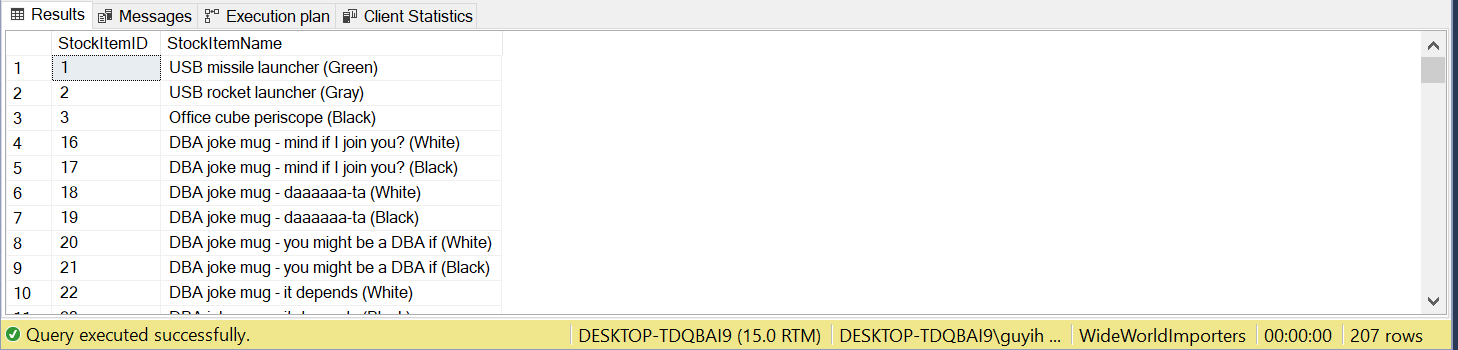


16.

SELECT StockItemID, StockItemName

FROM Warehouse.StockItems

WHERE JSON\_VALUE(CustomFields, '$.CountryOfManufacture') = 'China'



17.

SELECT JSON\_VALUE(s.CustomFields, '$.CountryOfManufacture') AS Country, SUM(ol.Quantity) AS Quantity

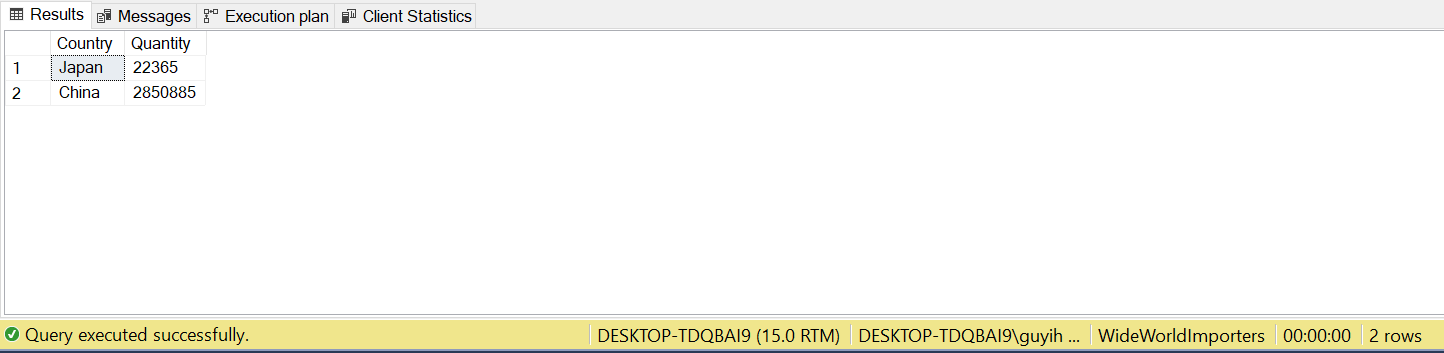
FROM sales.Orders o

JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID

JOIN Warehouse.StockItems s ON ol.StockItemID = s.StockItemID

WHERE YEAR(o.OrderDate) = 2015

GROUP BY JSON\_VALUE(s.CustomFields, '$.CountryOfManufacture')



18.

CREATE VIEW Sales.StockItemByYear AS

WITH cte0 AS (

SELECT StockGroupName, 2013 AS [Year]

FROM Warehouse.StockGroups

UNION ALL

SELECT StockGroupName, [Year] + 1

FROM cte0

WHERE [Year] < 2017

),

cte1 AS (

SELECT YEAR(o.OrderDate) AS [Year], sg.StockGroupName, SUM(ol.Quantity) AS Quantity

FROM Sales.Orders o

JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID

JOIN Warehouse.StockItems s ON ol.StockItemID =s.StockItemID

JOIN Warehouse.StockItemStockGroups g ON g.StockItemID = s.StockItemID

JOIN Warehouse.StockGroups sg ON g.StockGroupID = sg.StockGroupID

WHERE YEAR(o.OrderDate) BETWEEN 2013 AND 2017

GROUP BY YEAR(o.OrderDate), sg.StockGroupName

),

cte2 AS (

SELECT c0.StockGroupName, c0.[Year], ISNULL(c1.Quantity, 0) AS Quantity

FROM cte0 c0

LEFT JOIN cte1 c1 ON c0.[Year] = c1.[Year]

AND c0.StockGroupName = c1.StockGroupName

)

SELECT StockGroupName, [2013], [2014], [2015], [2016], [2017]

FROM cte2

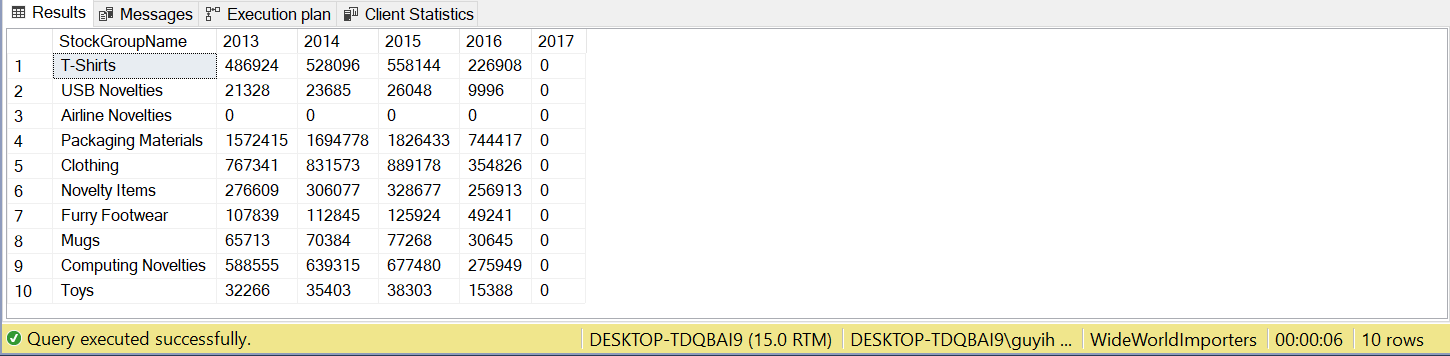
PIVOT

(

MIN(Quantity) FOR

Year IN ([2013], [2014], [2015], [2016], [2017])

) TBL



19.

CREATE VIEW Sales.StockItemByName AS

WITH cte0 AS (

SELECT StockGroupName, 2013 AS [Year]

FROM Warehouse.StockGroups

UNION ALL

SELECT StockGroupName, [Year] + 1

FROM cte0

WHERE [Year] < 2017

),

cte1 AS (

SELECT YEAR(o.OrderDate) AS [Year], sg.StockGroupName, SUM(ol.Quantity) AS Quantity

FROM Sales.Orders o

JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID

JOIN Warehouse.StockItems s ON ol.StockItemID =s.StockItemID

JOIN Warehouse.StockItemStockGroups g ON g.StockItemID = s.StockItemID

JOIN Warehouse.StockGroups sg ON g.StockGroupID = sg.StockGroupID

WHERE YEAR(o.OrderDate) BETWEEN 2013 AND 2017

GROUP BY YEAR(o.OrderDate), sg.StockGroupName

),

cte2 AS (

SELECT c0.StockGroupName, c0.[Year], ISNULL(c1.Quantity, 0) AS Quantity

FROM cte0 c0

LEFT JOIN cte1 c1 ON c0.[Year] = c1.[Year]

AND c0.StockGroupName = c1.StockGroupName

)

SELECT [Year], [Novelty Items], [Clothing], [Mugs], [T-Shirts],

[Airline Novelties], [Computing Novelties], [USB Novelties], [Furry Footwear], [Toys], [Packaging Materials]

FROM cte2

PIVOT

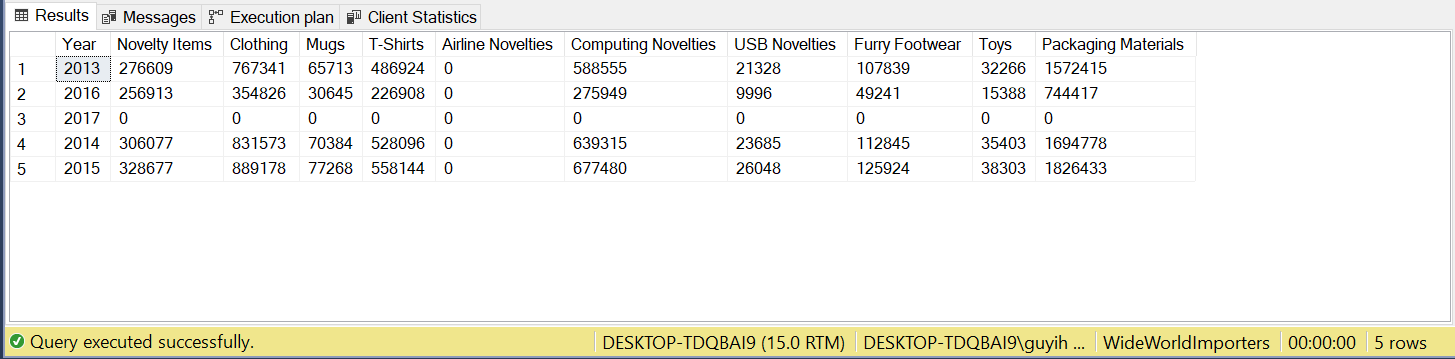
(

SUM(Quantity) FOR

StockGroupName IN ([Novelty Items], [Clothing], [Mugs], [T-Shirts],

[Airline Novelties], [Computing Novelties], [USB Novelties], [Furry Footwear], [Toys], [Packaging Materials])

) TBL



20.

CREATE FUNCTION Sales.OrderTotal (@orderid INT)

RETURNS TABLE

AS

RETURN (

SELECT OrderID, SUM(Quantity \* UnitPrice) AS Total

FROM Sales.OrderLines

WHERE OrderID = @orderid

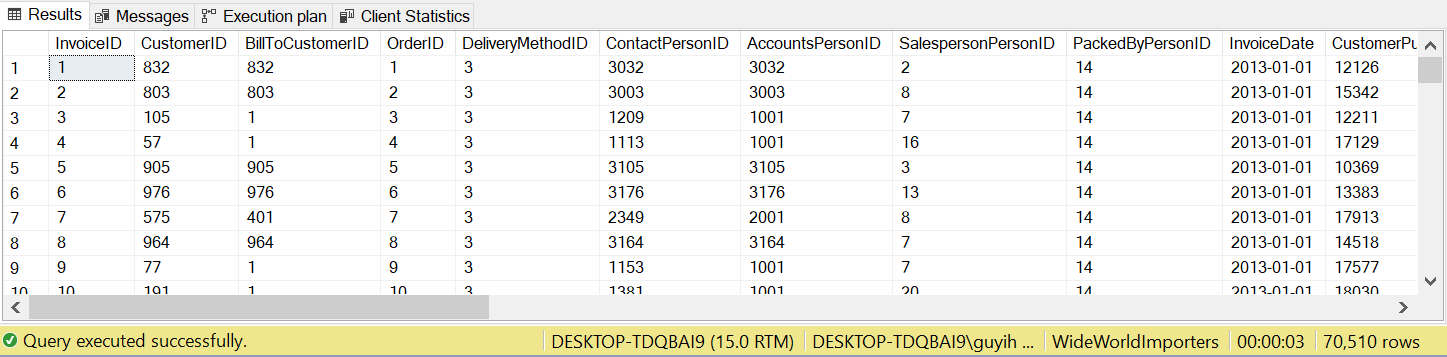
GROUP BY OrderID

)

SELECT \*

FROM Sales.Invoices i

CROSS APPLY Sales.OrderTotal(OrderID) f



21.

CREATE SCHEMA ods

GO

CREATE TABLE ods.Orders

(OrderID INT PRIMARY KEY,

OrderDate DATE,

OrderTotal DECIMAL(18, 2),

CustomerID INT)

GO

CREATE PROCEDURE ods.OrderTotalOfDate

@OrderDate DATE

AS

IF EXISTS (SELECT 1 FROM ods.Orders WHERE OrderDate = @OrderDate)

BEGIN

RAISERROR('Date Exists ', 16, 1)

END

ELSE

BEGIN

BEGIN TRANSACTION

INSERT INTO ods.Orders

SELECT o.OrderID, o.OrderDate, f.Total, o.CustomerID

FROM Sales.Orders o

CROSS APPLY Sales.OrderTotal(OrderID) f

WHERE o.OrderDate = @OrderDate

COMMIT

END

GO

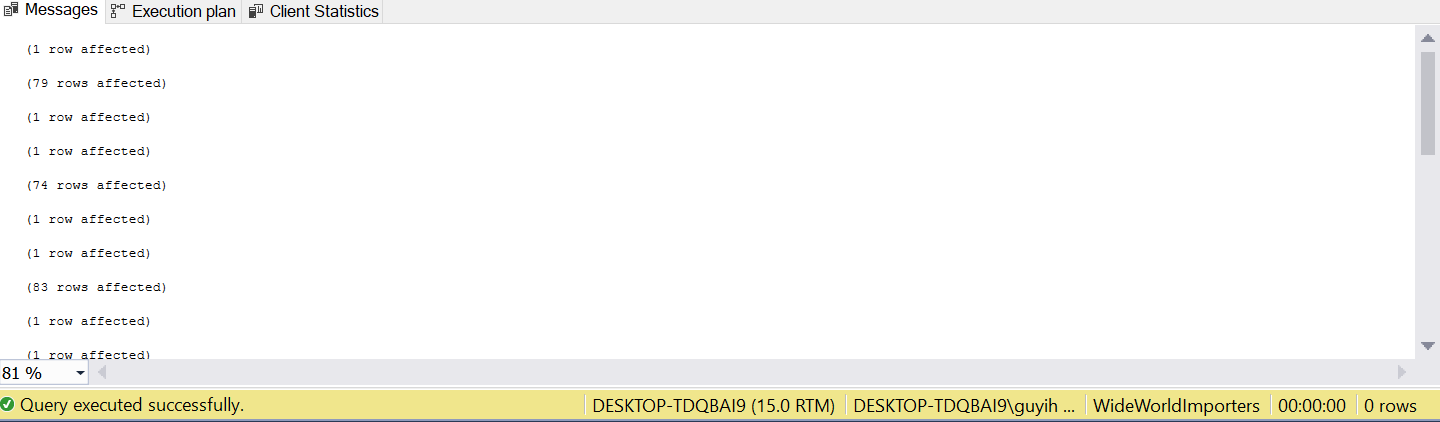
EXEC ods.OrderTotalOfDate '2013-01-01'

EXEC ods.OrderTotalOfDate '2013-01-02'

EXEC ods.OrderTotalOfDate '2013-01-03'

EXEC ods.OrderTotalOfDate '2013-01-04'

EXEC ods.OrderTotalOfDate '2013-01-05'



22.

CREATE TABLE ods.StockItems(

StockItemID INT PRIMARY KEY,

StockItemName NVARCHAR(100) NOT NULL,

SupplierID INT NOT NULL,

ColorID INT NULL,

UnitPackageID INT NOT NULL,

OuterPackageID INT NOT NULL,

Brand NVARCHAR(50) NULL,

Size NVARCHAR(20) NULL,

LeadTimeDays INT NOT NULL,

QuantityPerOuter INT NOT NULL,

IsChillerStock BIT NOT NULL,

Barcode NVARCHAR(50) NULL,

TaxRate DECIMAL(18, 3) NOT NULL,

UnitPrice DECIMAL(18, 2) NOT NULL,

RecommendedRetailPrice DECIMAL(18, 2) NULL,

TypicalWeightPerUnit DECIMAL(18, 3) NOT NULL,

MarketingComments NVARCHAR(MAX) NULL,

InternalComments NVARCHAR(MAX) NULL,

CountryOfManufacture NVARCHAR(20) NULL,

[Range] NVARCHAR(20) NULL,

Shelflife NVARCHAR(20) NULL

)

MERGE INTO ods.StockItems AS T

USING Warehouse.StockItems AS R

ON T.StockItemID = R.StockItemID

WHEN NOT MATCHED

THEN INSERT VALUES (R.StockItemID, R.StockItemName, R.SupplierID, R.ColorID,

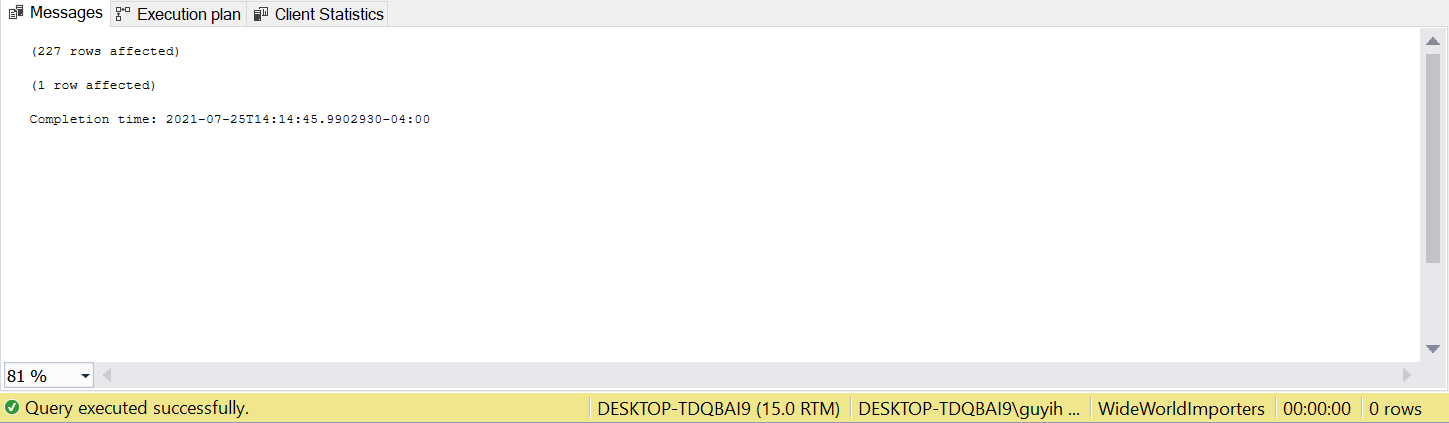
R.UnitPackageID, R.OuterPackageID, R.Brand, R.Size, R.LeadTimeDays,

R.QuantityPerOuter, R.IsChillerStock, R.Barcode, R.TaxRate, R.UnitPrice,

R.RecommendedRetailPrice, R.TypicalWeightPerUnit, R.MarketingComments,

R.InternalComments, JSON\_VALUE(R.CustomFields, '$.CountryOfManufacture'),

JSON\_VALUE(R.CustomFields, '$.Range'), JSON\_VALUE(R.CustomFields, '$.ShelfLife'));



23.

CREATE PROCEDURE ods.NewOrderTotalOfDate

@OrderDate DATE

AS

BEGIN TRANSACTION

DELETE FROM ods.Orders

WHERE OrderDate < @OrderDate

COMMIT

BEGIN TRANSACTION

MERGE ods.Orders T

USING (

SELECT o.OrderID, o.OrderDate, f.Total, o.CustomerID

FROM Sales.Orders o

CROSS APPLY Sales.OrderTotal(OrderID) f

WHERE DATEDIFF(d, @OrderDate, OrderDate) BETWEEN 1 AND 7

) R

ON T.OrderID = R.OrderID

WHEN NOT MATCHED

THEN INSERT VALUES (R.OrderID, R.OrderDate, R.Total, R.CustomerID);

COMMIT

24.

DECLARE @json NVARCHAR(MAX) = N'{

"PurchaseOrders":[

{

"StockItemName":"Panzer Video Game",

"Supplier":"7",

"UnitPackageId":"1",

"OuterPackageId":[

6,

7

],

"Brand":"EA Sports",

"LeadTimeDays":"5",

"QuantityPerOuter":"1",

"TaxRate":"6",

"UnitPrice":"59.99",

"RecommendedRetailPrice":"69.99",

"TypicalWeightPerUnit":"0.5",

"CountryOfManufacture":"Canada",

"Range":"Adult",

"OrderDate":"2018-01-01",

"DeliveryMethod":"Post",

"ExpectedDeliveryDate":"2018-02-02",

"SupplierReference":"WWI2308"

},

{

"StockItemName":"Panzer Video Game",

"Supplier":"5",

"UnitPackageId":"1",

"OuterPackageId":"7",

"Brand":"EA Sports",

"LeadTimeDays":"5",

"QuantityPerOuter":"1",

"TaxRate":"6",

"UnitPrice":"59.99",

"RecommendedRetailPrice":"69.99",

"TypicalWeightPerUnit":"0.5",

"CountryOfManufacture":"Canada",

"Range":"Adult",

"OrderDate":"2018-01-025",

"DeliveryMethod":"Post",

"ExpectedDeliveryDate":"2018-02-02",

"SupplierReference":"269622390"

}

]

}'

WITH cte AS (

(SELECT \*

FROM OPENJSON(@json, '$.PurchaseOrders')

WITH (

StockItemName NVARCHAR(50),

Supplier INT,

UnitPackageId INT,

OuterPackageId NVARCHAR(MAX) AS JSON,

Brand NVARCHAR(20),

LeadTimeDays INT,

QuantityPerOuter INT,

TaxRate DECIMAL(18, 3),

UnitPrice DECIMAL(18, 2),

RecommendedRetailPrice DECIMAL(18, 2),

TypicalWeightPerUnit DECIMAL(18, 3),

CountryOfManufacture NVARCHAR(50),

Range NVARCHAR(20),

OrderDate NVARCHAR(20),

DeliveryMethod NVARCHAR(20),

ExpectedDeliveryDate NVARCHAR(20),

SupplierReference NVARCHAR(20)

)

CROSS APPLY OPENJSON(OuterPackageId) WITH (NewOuterPackageId INT '$')

)

UNION ALL

(SELECT \*

FROM OPENJSON(@json, '$.PurchaseOrders')

WITH (

StockItemName NVARCHAR(50),

Supplier INT,

UnitPackageId INT,

OuterPackageId NVARCHAR(MAX),

Brand NVARCHAR(20),

LeadTimeDays INT,

QuantityPerOuter INT,

TaxRate DECIMAL(18, 3),

UnitPrice DECIMAL(18, 2),

RecommendedRetailPrice DECIMAL(18, 2),

TypicalWeightPerUnit DECIMAL(18, 3),

CountryOfManufacture NVARCHAR(50),

Range NVARCHAR(20),

OrderDate NVARCHAR(20),

DeliveryMethod NVARCHAR(20),

ExpectedDeliveryDate NVARCHAR(20),

SupplierReference NVARCHAR(20),

OuterPackageId INT

)

)

)

DECLARE @maxid INT = (SELECT MAX(StockItemID)

FROM Warehouse.StockItems);

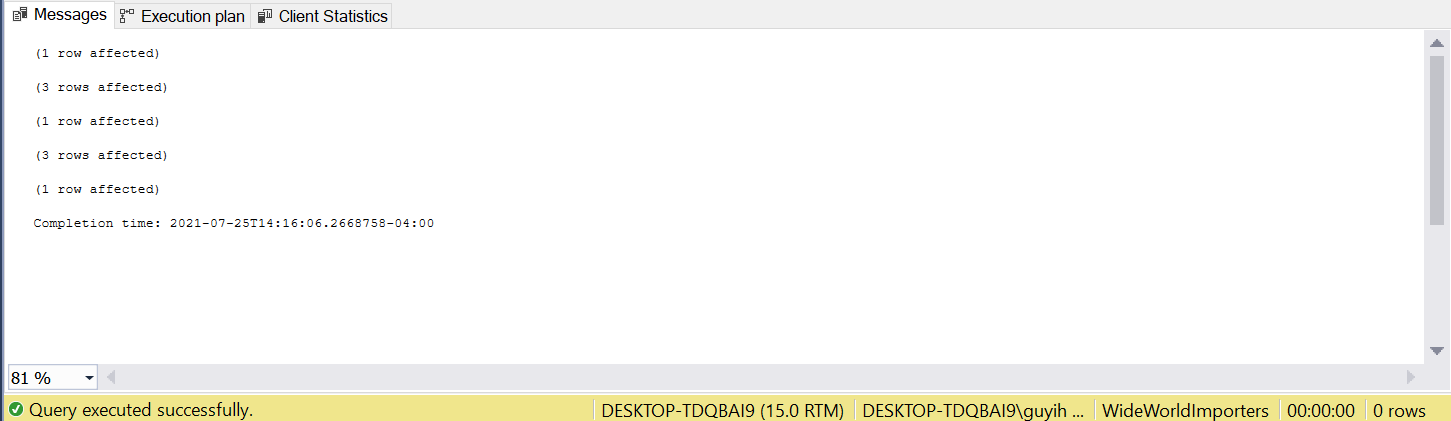
SELECT IDENTITY(INT, 1, 1) AS StockItemId, \* INTO #stock

FROM cte WHERE OuterPackageId is NOT NULL

INSERT INTO Warehouse.StockItems (StockItemID, StockItemName, SupplierID, UnitPackageId, OuterPackageId, Brand, LeadTimeDays, QuantityPerOuter, IsChillerStock, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, LastEditedBy)

SELECT StockItemId + @maxid, StockItemName + '(' + CAST(StockItemID AS NVARCHAR) + ')' , Supplier, UnitPackageId, NewOuterPackageId, Brand, LeadTimeDays, QuantityPerOuter, 0, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, 1

FROM #stock



25 .

SELECT Year AS Year,

[Novelty Items] AS 'StockGroup.Novelty Items',

[Clothing] AS 'StockGroup.Clothing',

[Mugs] AS 'StockGroup.Mugs',

[T-Shirts] AS 'StockGroup.T-Shirts',

[Airline Novelties] AS 'StockGroup.Airline Novelties',

[Computing Novelties] AS 'StockGroup.Computing Novelties',

[USB Novelties] AS 'StockGroup.USB Novelties',

[Furry Footwear] AS 'StockGroup.Furry Footwear',

[Toys] AS 'StockGroup.Toys',

[Packaging Materials] AS 'StockGroup.Packaging Materials'

FROM Sales.StockItemByName

FOR JSON PATH

[{"Year":2013,"StockGroup":{"Novelty Items":276609,"Clothing":767341,"Mugs":65713,"T-Shirts":486924,"Airline Novelties":0,"Computing Novelties":588555,"USB Novelties":21328,"Furry Footwear":107839,"Toys":32266,"Packaging Materials":1572415}},{"Year":2016,"StockGroup":{"Novelty Items":256913,"Clothing":354826,"Mugs":30645,"T-Shirts":226908,"Airline Novelties":0,"Computing Novelties":275949,"USB Novelties":9996,"Furry Footwear":49241,"Toys":15388,"Packaging Materials":744417}},{"Year":2017,"StockGroup":{"Novelty Items":0,"Clothing":0,"Mugs":0,"T-Shirts":0,"Airline Novelties":0,"Computing Novelties":0,"USB Novelties":0,"Furry Footwear":0,"Toys":0,"Packaging Materials":0}},{"Year":2014,"StockGroup":{"Novelty Items":306077,"Clothing":831573,"Mugs":70384,"T-Shirts":528096,"Airline Novelties":0,"Computing Novelties":639315,"USB Novelties":23685,"Furry Footwear":112845,"Toys":35403,"Packaging Materials":1694778}},{"Year":2015,"StockGroup":{"Novelty Items":328677,"Clothing":889178,"Mugs":77268,"T-Shirts":558144,"Airline Novelties":0,"Computing Novelties":677480,"USB Novelties":26048,"Furry Footwear":125924,"Toys":38303,"Packaging Materials":1826433}}]

26.

SELECT Year AS '@Year',

[Novelty Items] AS NoveltyItems,

[Clothing],

[Mugs],

[T-Shirts],

[Airline Novelties] AS AirlineNovelties,

[Computing Novelties] AS ComputingNovelties,

[USB Novelties] AS USBNovelties,

[Furry Footwear] AS FurryFootwear,

[Toys],

[Packaging Materials] AS PackagingMaterials

FROM Sales.StockItemByName

FOR XML PATH('StockItems')

<StockItems Year="2013">

<NoveltyItems>276609</NoveltyItems>

<Clothing>767341</Clothing>

<Mugs>65713</Mugs>

<T-Shirts>486924</T-Shirts>

<AirlineNovelties>0</AirlineNovelties>

<ComputingNovelties>588555</ComputingNovelties>

<USBNovelties>21328</USBNovelties>

<FurryFootwear>107839</FurryFootwear>

<Toys>32266</Toys>

<PackagingMaterials>1572415</PackagingMaterials>

</StockItems>

<StockItems Year="2016">

<NoveltyItems>256913</NoveltyItems>

<Clothing>354826</Clothing>

<Mugs>30645</Mugs>

<T-Shirts>226908</T-Shirts>

<AirlineNovelties>0</AirlineNovelties>

<ComputingNovelties>275949</ComputingNovelties>

<USBNovelties>9996</USBNovelties>

<FurryFootwear>49241</FurryFootwear>

<Toys>15388</Toys>

<PackagingMaterials>744417</PackagingMaterials>

</StockItems>

<StockItems Year="2017">

<NoveltyItems>0</NoveltyItems>

<Clothing>0</Clothing>

<Mugs>0</Mugs>

<T-Shirts>0</T-Shirts>

<AirlineNovelties>0</AirlineNovelties>

<ComputingNovelties>0</ComputingNovelties>

<USBNovelties>0</USBNovelties>

<FurryFootwear>0</FurryFootwear>

<Toys>0</Toys>

<PackagingMaterials>0</PackagingMaterials>

</StockItems>

<StockItems Year="2014">

<NoveltyItems>306077</NoveltyItems>

<Clothing>831573</Clothing>

<Mugs>70384</Mugs>

<T-Shirts>528096</T-Shirts>

<AirlineNovelties>0</AirlineNovelties>

<ComputingNovelties>639315</ComputingNovelties>

<USBNovelties>23685</USBNovelties>

<FurryFootwear>112845</FurryFootwear>

<Toys>35403</Toys>

<PackagingMaterials>1694778</PackagingMaterials>

</StockItems>

<StockItems Year="2015">

<NoveltyItems>328677</NoveltyItems>

<Clothing>889178</Clothing>

<Mugs>77268</Mugs>

<T-Shirts>558144</T-Shirts>

<AirlineNovelties>0</AirlineNovelties>

<ComputingNovelties>677480</ComputingNovelties>

<USBNovelties>26048</USBNovelties>

<FurryFootwear>125924</FurryFootwear>

<Toys>38303</Toys>

<PackagingMaterials>1826433</PackagingMaterials>

</StockItems>

27.

CREATE TABLE ods.ConfirmedDeviveryJson

(ID INT PRIMARY KEY,

Date DATE,

Value NVARCHAR(MAX))

GO

CREATE PROCEDURE ods.InsertInvoicesOfDate

@Date DATE

AS

BEGIN

DECLARE @i NVARCHAR(MAX) = '';

DECLARE @oi NVARCHAR(MAX) = '';

DECLARE @column NVARCHAR(MAX);

DECLARE @query NVARCHAR(MAX);

DECLARE @json NVARCHAR(MAX);

SELECT @i = @i + 'i.' + COLUMN\_NAME + ', '

FROM (SELECT COLUMN\_NAME

FROM(

SELECT COLUMN\_NAME

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = N'Invoices') a

WHERE COLUMN\_NAME != 'InvoiceDate'

) b

SELECT @oi = @oi + 'oi.' + COLUMN\_NAME + ', '

FROM (

SELECT COLUMN\_NAME

FROM(

SELECT COLUMN\_NAME

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'InvoiceLines') a

WHERE COLUMN\_NAME NOT IN ('InvoiceID', 'LastEditedBy', 'LastEditedWhen')

) b

SET @column = @i + SUBSTRING(@oi, 0, LEN(@oi))

SET @query = 'SET @json = (SELECT ID = oi1.InvoiceLineID, Date = i1.InvoiceDate,

Value = (

SELECT ' + @column + '

FROM Sales.Invoices i

JOIN Sales.InvoiceLines oi ON i.InvoiceID = oi.InvoiceID

WHERE oi.InvoiceLineID = oi1.InvoiceLineID

FOR JSON PATH, ROOT(''Value''), INCLUDE\_NULL\_VALUES)

FROM Sales.Invoices i1

JOIN Sales.InvoiceLines oi1 ON i1.InvoiceID = oi1.InvoiceID

WHERE i1.InvoiceDate = @Date

FOR JSON PATH, INCLUDE\_NULL\_VALUES)'

EXEC SP\_EXECUTESQL @query, N'@Date DATE, @json NVARCHAR(MAX) OUT', @Date, @json OUT

INSERT INTO ods.ConfirmedDeviveryJson

SELECT \*

FROM OPENJSON(@json)

WITH(

ID INT '$.ID',

Date DATE '$.Date',

Value NVARCHAR(MAX) '$.Value' AS JSON)

END

GO

DECLARE @loop NVARCHAR(MAX) = ''

SELECT @loop = @loop + N'EXEC ods.InsertInvoicesOfDate ''' + CAST(InvoiceDate AS NVARCHAR)+ '''; '

FROM (SELECT DISTINCT InvoiceDate

FROM Sales.Invoices

WHERE CustomerID = 1) a

EXEC SP\_EXECUTESQL @loop

28.

A transaction is a set of statements, like some DML, DDL operations. One transaction has two outcomes, one is COMMIT and another is ROLLBACK. If one transaction is committed without any error, all modifications done by this transaction are saved in the database permanently. If some errors happen during the commit of transaction, SQL Server will automatically rollback to the beginning of the transaction. So these modifications will not be saved. A reliable database should follow a principle called ACID, which works directly on transaction. ACID represents atomicity, consistency, isolation, and durability. Atomicity means that one transaction can only fail or succeed. No partial transaction is committed. Consistency means that any transaction must make sure the integrity of data. Isolation means that two transactions should not affect each other. This principle is determined by isolation level. Durability means that any modification is saved permanently in the database if one transaction is committed successfully.

When it comes to isolation in ACID, concurrency problems should be introduced. Concurrency occurs when two or more transactions are trying to access or change same object of database at the same time. Because there is no guarantee that which transaction will succeed during concurrency, it will cause four main concurrency problems. They are lost update, dirty read, non-repeatable read and phantom read. Lost update occurs when one transaction is trying to modify some data from other uncommitted transactions. If some other transactions rollback, modifications done by first transaction will lose. Dirty read occurs when one transaction reads data several times from other uncommitted transactions. If some other transactions rollback, the first transaction will read invalid data. Non-repeatable read occurs when one transaction is reading data several times and other transactions are committing modifications at the same time. This will cause that those different results are read by first transaction. Phantom read occurs when one transaction is reading data several times and other transactions are committing insert operation at the same time. This will cause that redundant data are read from first transaction.

Isolation levels are designed to solve those concurrency problems above. There are four isolation levels, and they are read uncommitted, read committed, repeatable read and serializable. Read uncommitted can only solve lost update. Read committed can solve more one, which is dirty read compared to read uncommitted. Repeatable read obviously solve more one called non-repeatable read compared to read committed. Serializable can solve all these concurrency problems. From read uncommitted to serializable level, we will suffer less from concurrency effects but with more bad efficiency of concurrency.

Lock is a strategy of database to implement these four isolation levels. There seven lock modes in SQL Server. They are shared lock, update lock, exclusive lock, intent lock, schema lock, bulk update lock and key-range lock. Shared locks allow multiple transactions to read a resource but not do modifications. Update locks is to avoid deadlock. Exclusive locks prevent any other transactions from accessing some data. Intent locks to protect placing a shared lock or exclusive lock on a resource lower in the lock hierarchy. Schema locks are used during DDL operation. Bulk update locks allow multiple threads to bulk load data concurrently into the same table while preventing other processes that are not bulk loading data from accessing the table. Key-range locks are used to protect the range of rows of table while setting serializable isolation level.

29.

I will firstly try to figure out what errors occurred during the weekend. Depending on the SQL Server Error Logs, I will then try to group these related errors based on their error codes. From easy-level to hard-level, I will try to fix them one group by one group. For some root errors probably caused by system designs, I will try to discuss with my co-workers or senior level developers. These errors are more complex and difficult to fix them. More departments may need to collaborate to figure the root of this error and fix it.

For slow response speed, I also need to filter what causes huge costs of system based on event logs. These may be caused by bad database designs, bad query designs and bad concurrency designs. For database designs, I need to check related constraints, like index designs. For query designs, I need to optimize the query performance by system providing tools, like Tuning Advisor. I also may need to create more stored procedures and functions to increase the efficiency. For concurrency problems, I will try to check whether there are deadlocks to make the system stuck. Bad locks designed are needed to be removed to improve concurrency performance.

30.

To Create Maintenance Plans:

First, execute the query below to enable Agent XPs Server Configuration Option.

sp\_configure 'show advanced options', 1;

GO

RECONFIGURE;

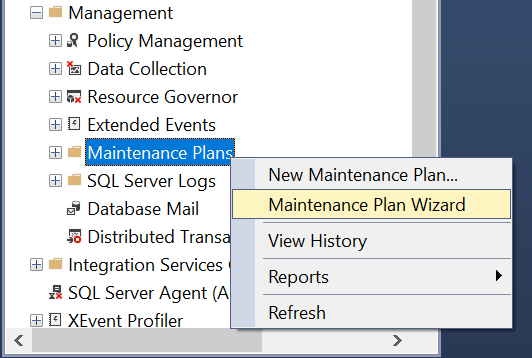
GO

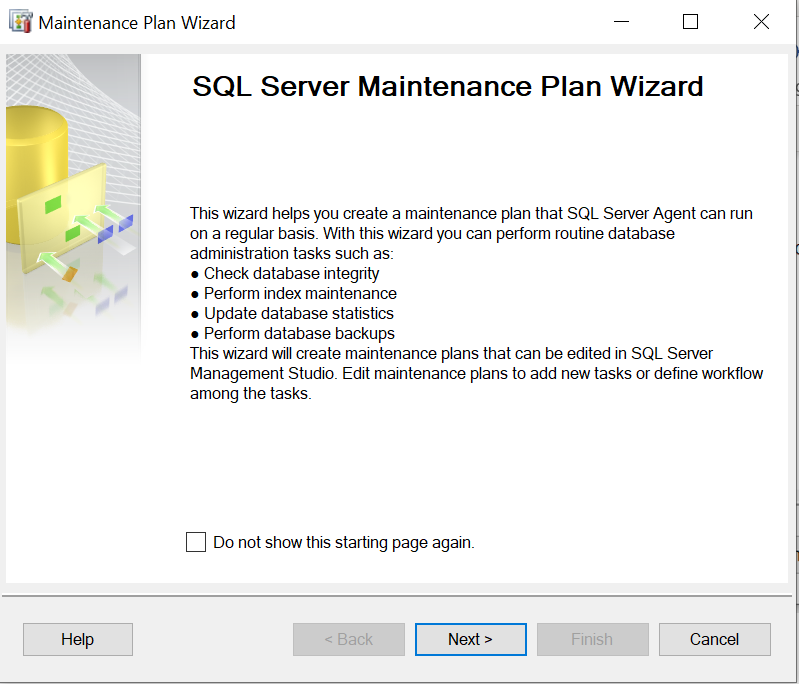
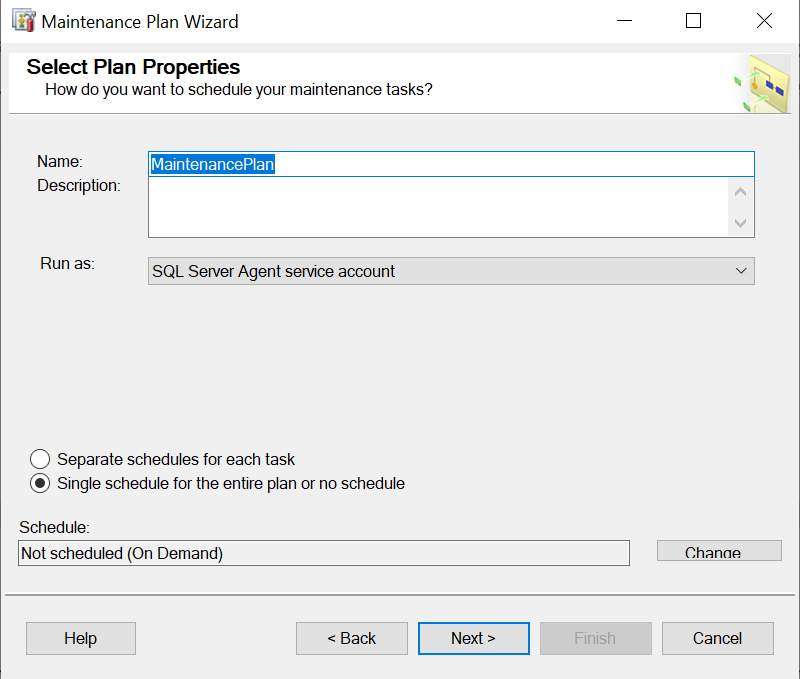
sp\_configure 'Agent XPs', 1;

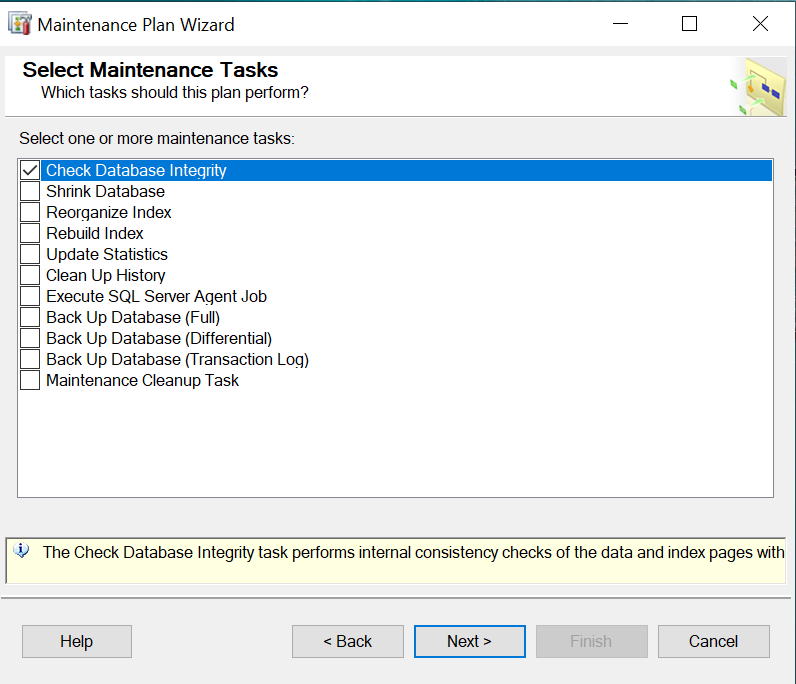
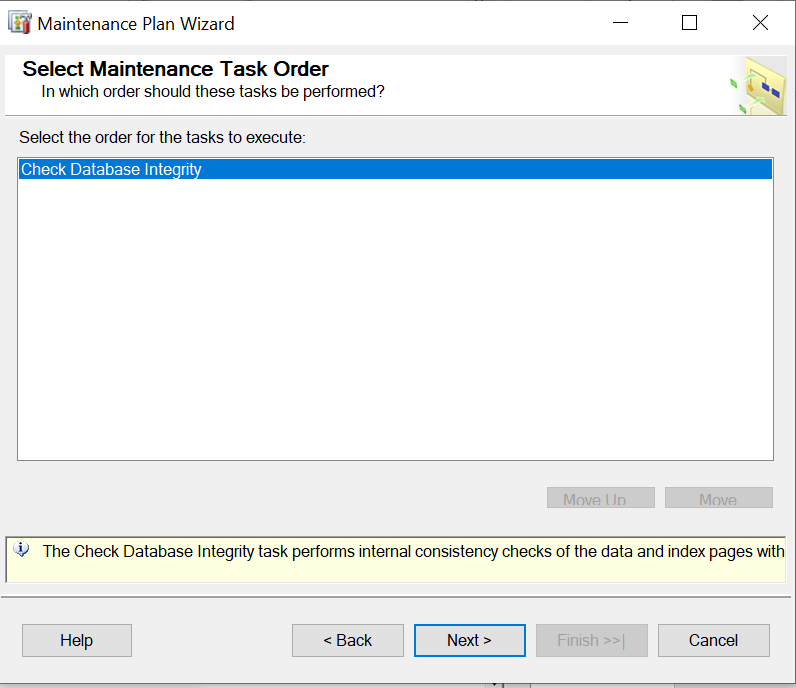
GO

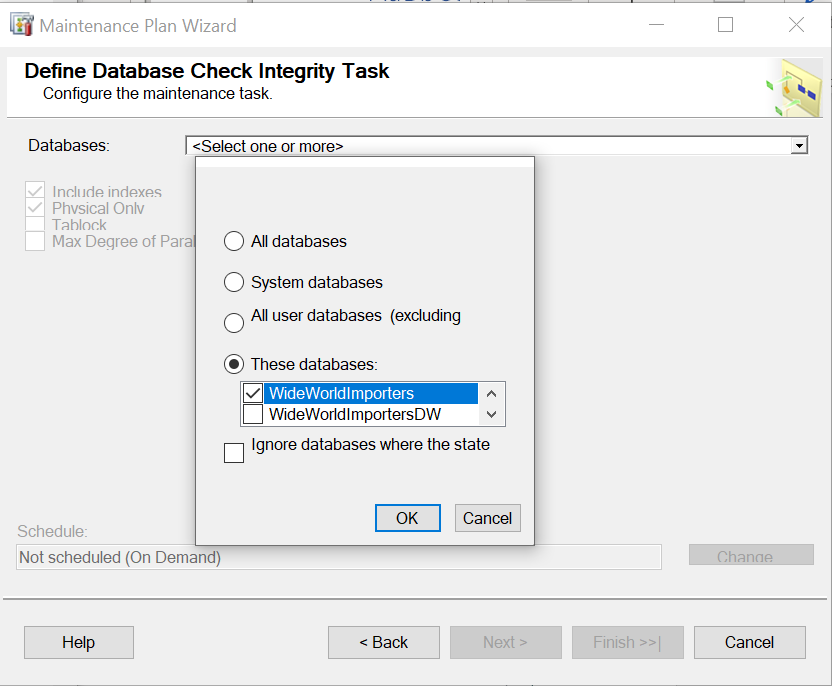
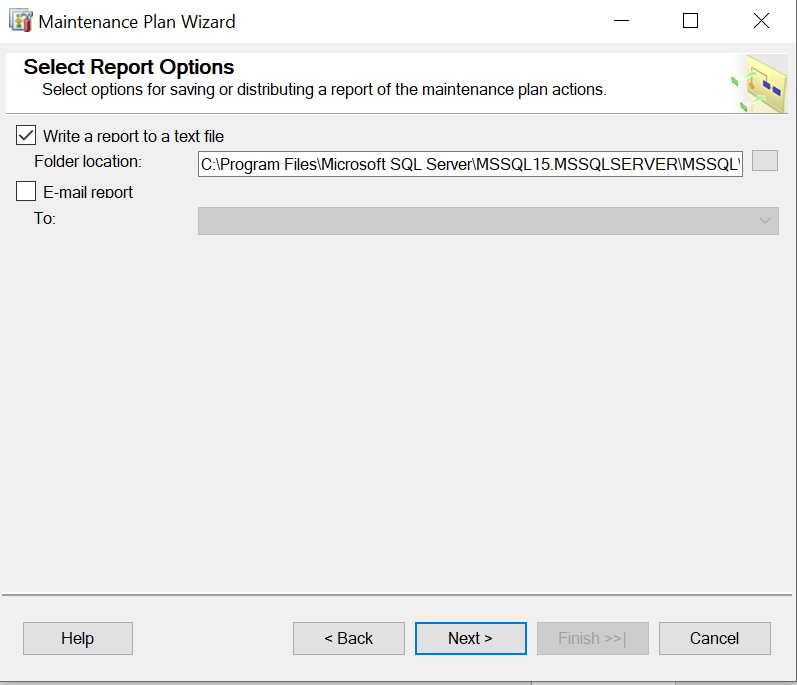
RECONFIGURE

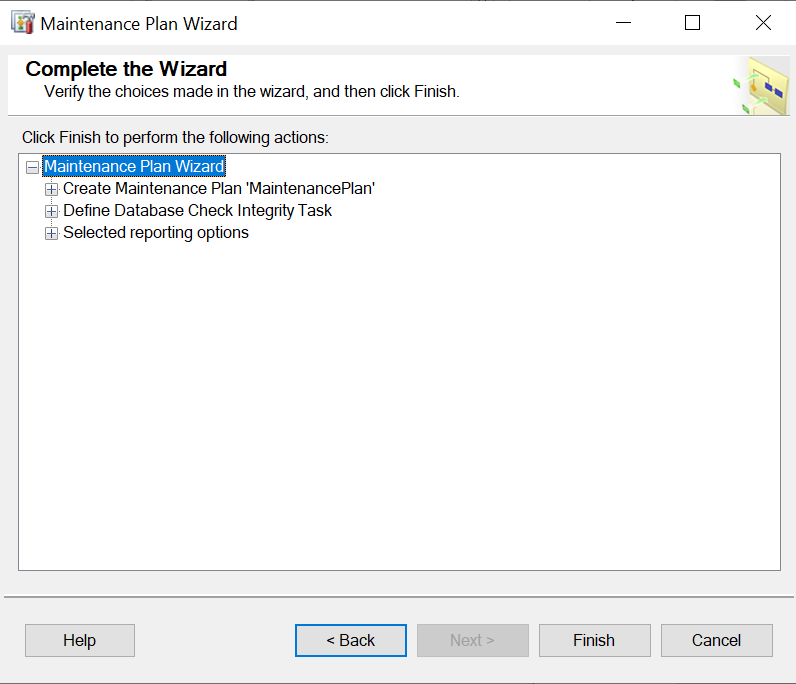
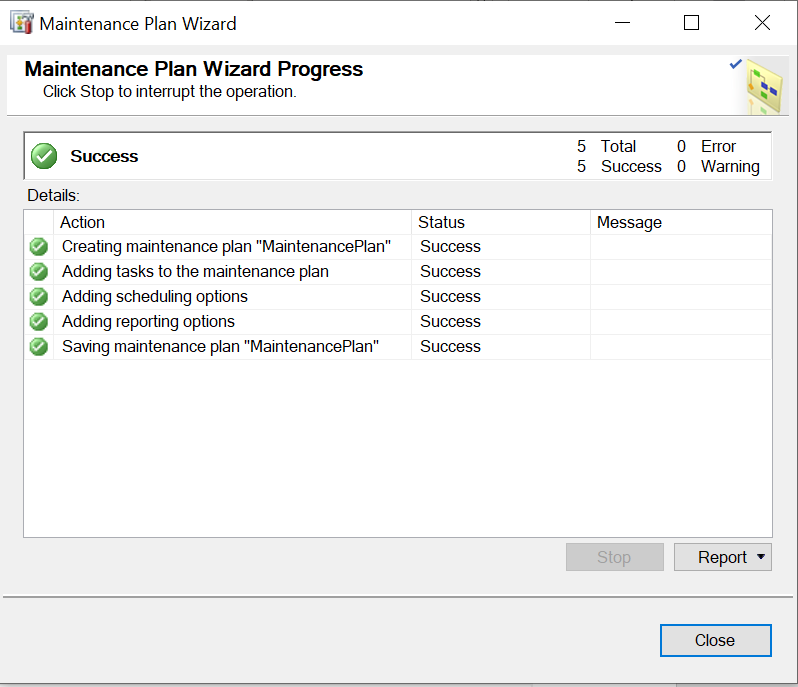
GO



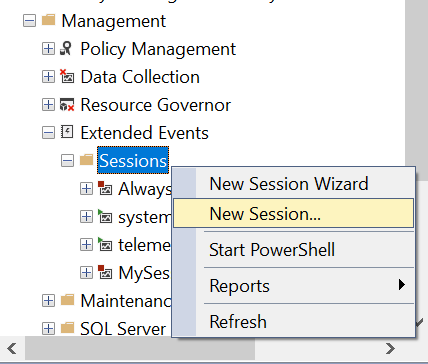
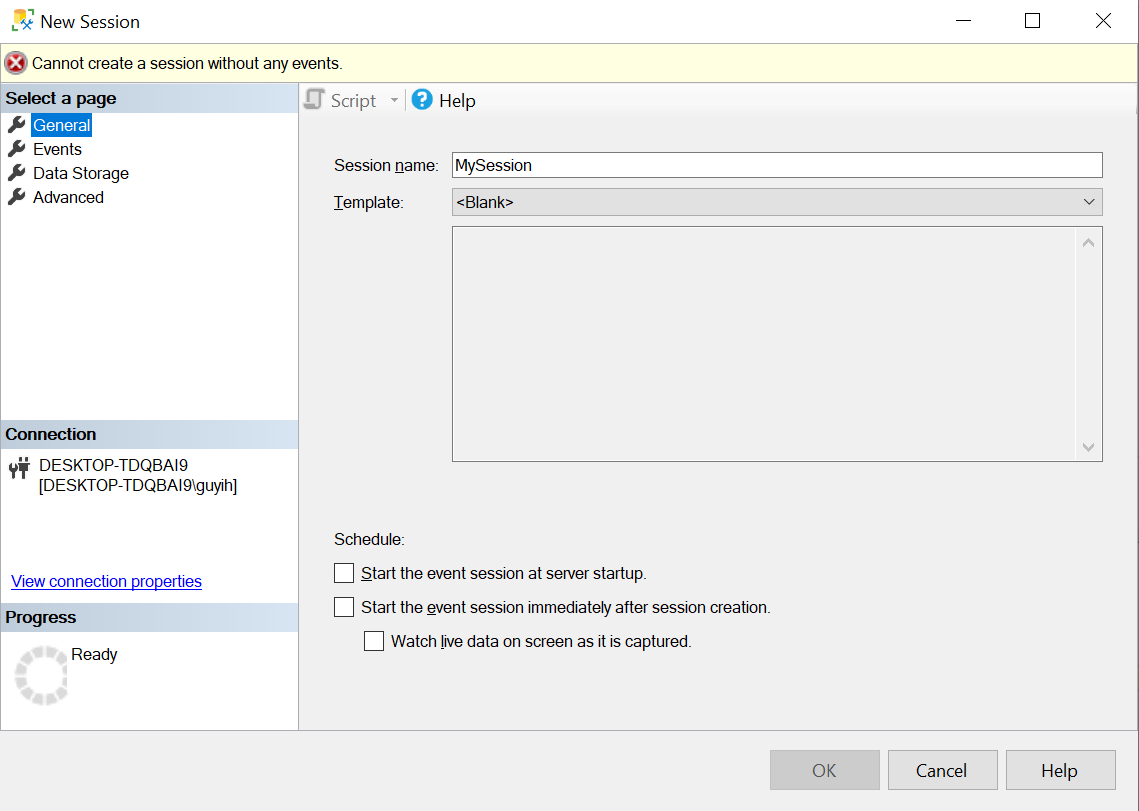
 

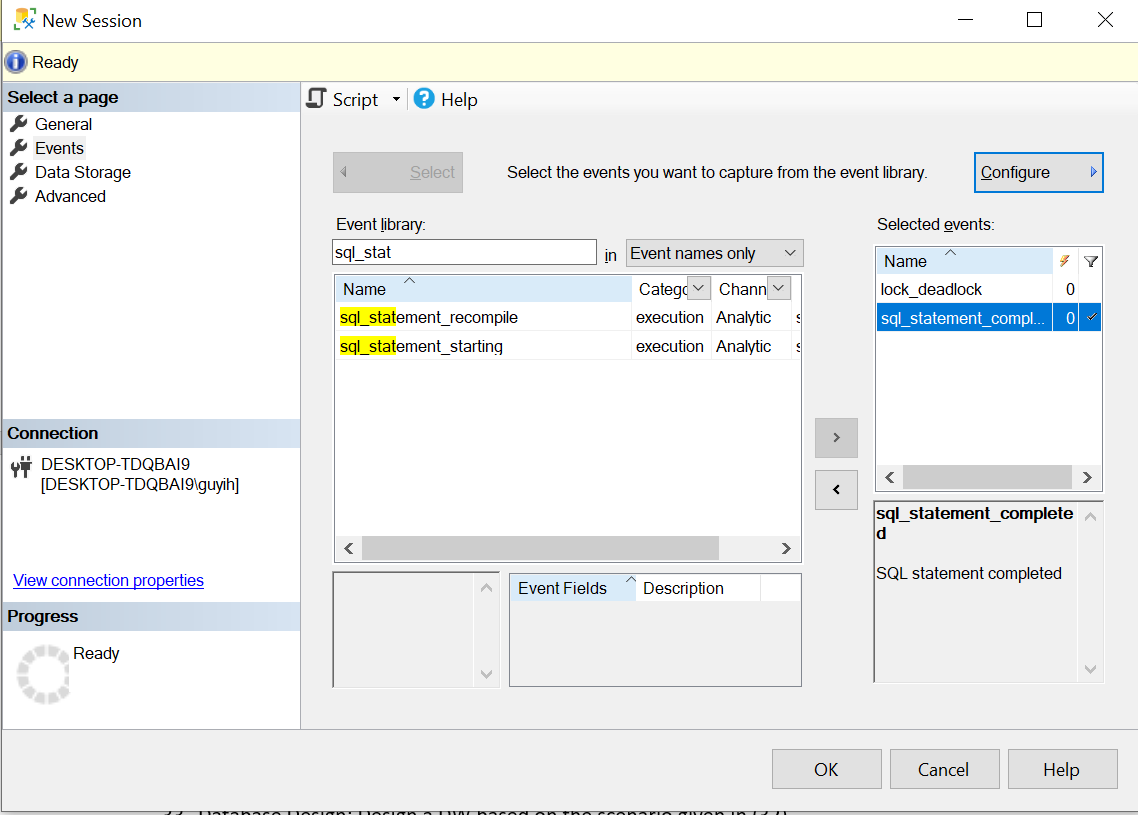
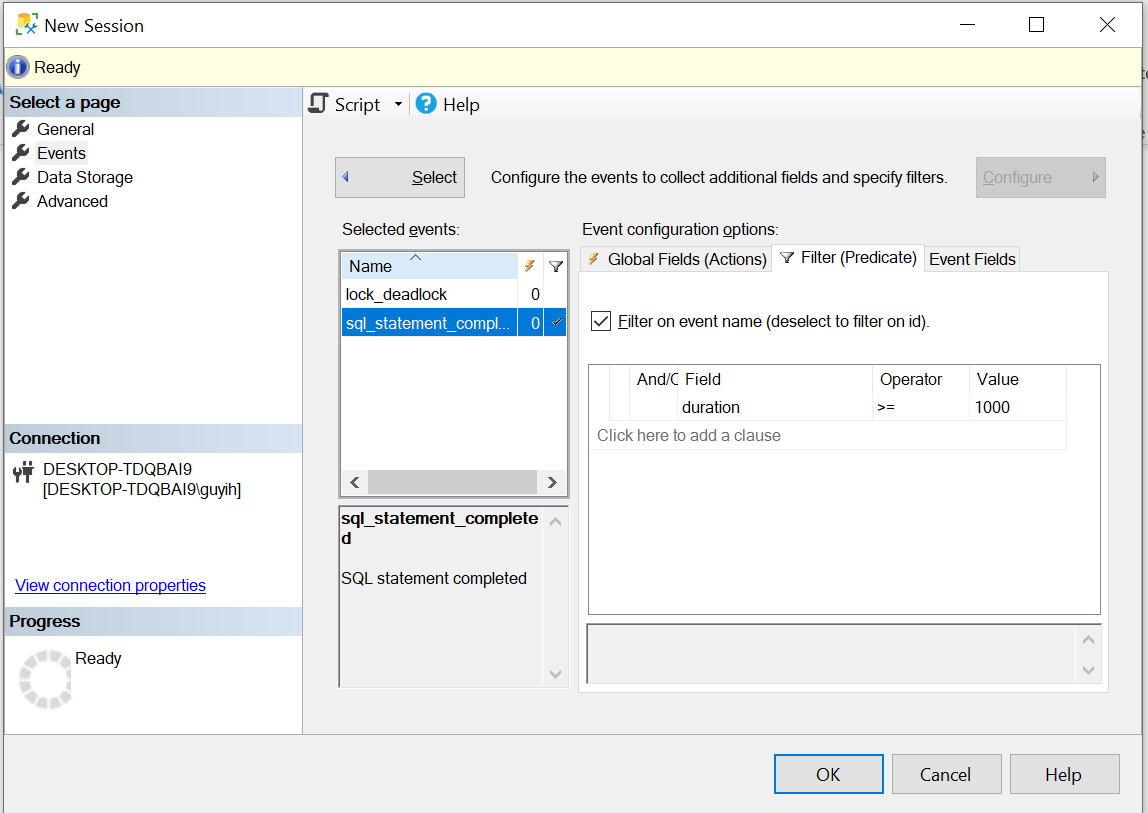
 

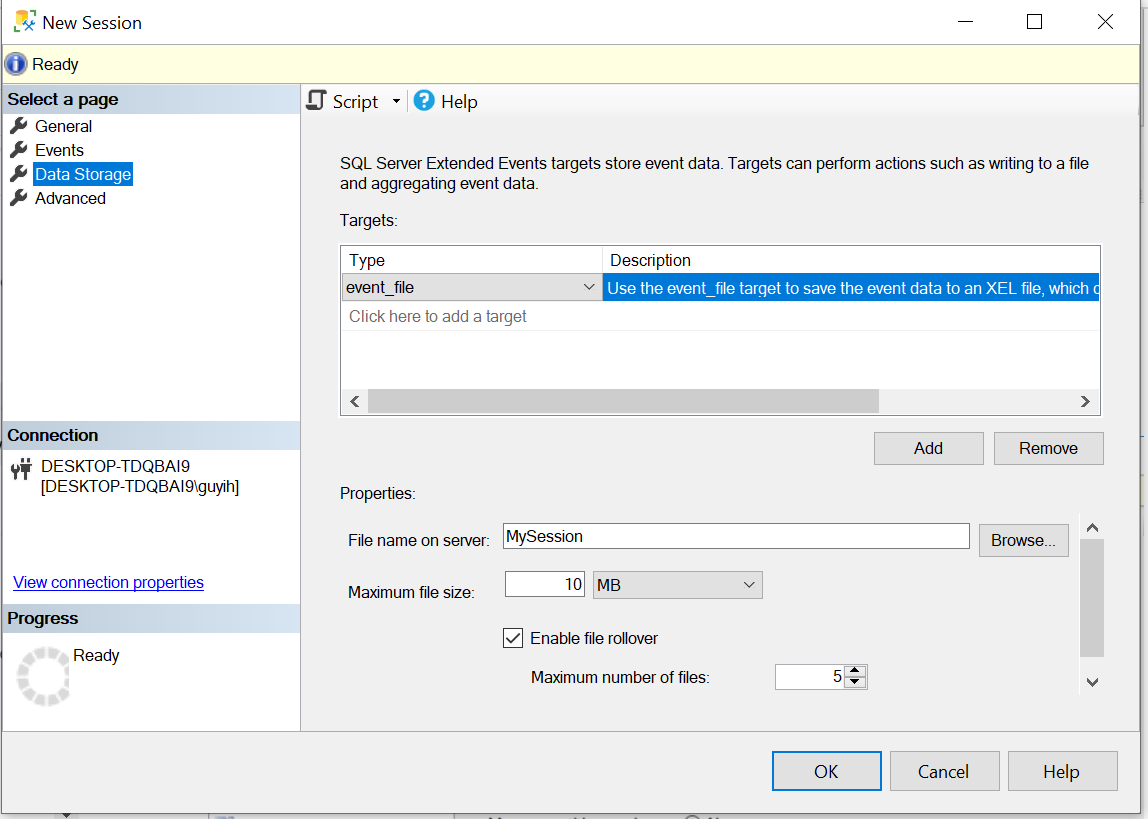
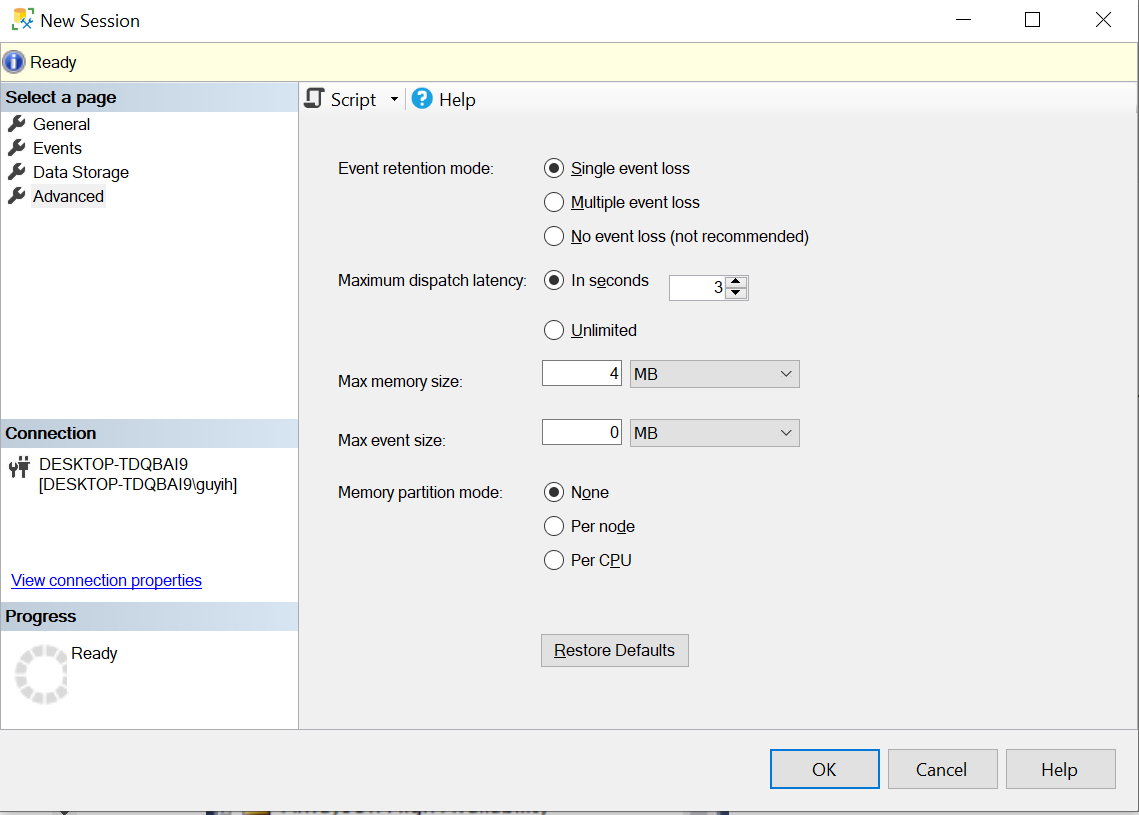
 

To Create Extended Events:

Do not click OK button but click upper-left options

Finally click OK button, extended events will be created.

Group Assignment:

31.

For merging Logon and Person information into WorldWideImporters database, these two parts of information can be merged into only one table called Application.People. Firstly, in order to make sure the data integrity of primary key constraint, We need to DECLARE a variable called @maxid that stores the maximum PersonID and then add this maximum value to the BussinessEntityID. This operation makes sure that we will not violate the primary key constraint. Since names in Adventure Works database are split into first names and last names, we need to combine first and last name together to make sure we can match it to Application.People. For IsPermittedToLogon, IsExternalLogonProvider and IsSystemUser information that is not allowed to be NULL and can not be found in Adventures Works database, we set all to 0. For PassWordHash information, we convert it to VARBINARY data type to match it to main database. For IsEmployee and IsSalesperson information, we use two LEFT JOIN operation to check whether it is 1 or 0. Because some PhoneNumber information has different data structures, we use SUBSTRING function to extract same part.

For the part of product table in WideWorldImporters, first we have to an overall look at columns, focusing on whether a column is the primary key or foreign key or not null. The fact is that we have two unique id columns for product with product id and supplier id respectively. Since the corresponding id columns for products in AdventureWorks2019 are assigned for another line of ids, we must reassign the id column in product table in AdventureWorks2019 as a different series of id to avoid merge conflict. The rule is that we want to assign the id for new products from the new company continuing from the maximum id number of datasets in old system. Next phase is to find all columns with not null constraints because when we insert the new data in the old columns, we do not want the new data makes the conflict. The related columns involve StockItemName and LeadTimeDays, etc. However, sometimes we cannot find the corresponding column in the new table and the temporary solution is to assign a value to represent that we cannot find the value. For int of float value, we assign -1 because -1 cannot be used for price or quantity sold. For strings, we usually assign ‘Unknown’ to distinguish with other known values.

Another issue our group notice is that supplier id is a foreign key in product table in WideWorldImporters. Therefore, we also assign new id for these new suppliers in the original supplier table in WideWorldImporters. After we grabbed all necessary and available data from new company dataset, we saved all these data into a temporary table and did the final check for all data types and constraints. The following is an example for the temp table.



Finally the data of product from AdventureWorks2019 was successfully merged into WideWorldImporters. The following part is the whole query we used.

DECLARE @maxid INT;

DECLARE @col NVARCHAR(MAX) = '';

DECLARE @query NVARCHAR(MAX);

SELECT @maxid = MAX(personID)

FROM Application.People

SELECT p.BusinessEntityID + @maxid AS PersonID,

p.FirstName + ' ' + p.LastName AS FullName, p.FirstName AS PreferredName,

0 AS IsPermittedToLogon, e.EmailAddress AS LogonName,

0 AS IsExternalLogonProvider, CONVERT(VARBINARY, pa.PasswordHash) AS HashedPassword,

0 AS IsSystemUser,

CASE WHEN em.BusinessEntityID IS NOT NULL THEN 1 ELSE 0 END AS IsEmployee,

CASE WHEN s.BusinessEntityID IS NOT NULL THEN 1 ELSE 0 END AS IsSalesperson,

'(' + LEFT(RIGHT(ph.PhoneNumber, 12), 3) + ') ' + RIGHT(ph.PhoneNumber, 8) AS PhoneNumber,

e.EmailAddress AS EmailAddress, 1 AS LastEditedBy

INTO #person

FROM AdventureWorks2019.Person.Person p

JOIN AdventureWorks2019.Person.EmailAddress e ON p.BusinessEntityID = e.BusinessEntityID

JOIN AdventureWorks2019.Person.[Password] pa ON p.BusinessEntityID = pa.BusinessEntityID

LEFT JOIN AdventureWorks2019.HumanResources.Employee em ON p.BusinessEntityID = em.BusinessEntityID

LEFT JOIN AdventureWorks2019.Sales.SalesPerson s ON p.BusinessEntityID = s.BusinessEntityID

JOIN AdventureWorks2019.Person.PersonPhone ph ON p.BusinessEntityID = ph.BusinessEntityID

SELECT @col = @col + name +','

FROM tempdb.sys.columns

WHERE OBJECT\_ID = OBJECT\_ID('tempdb..#person')

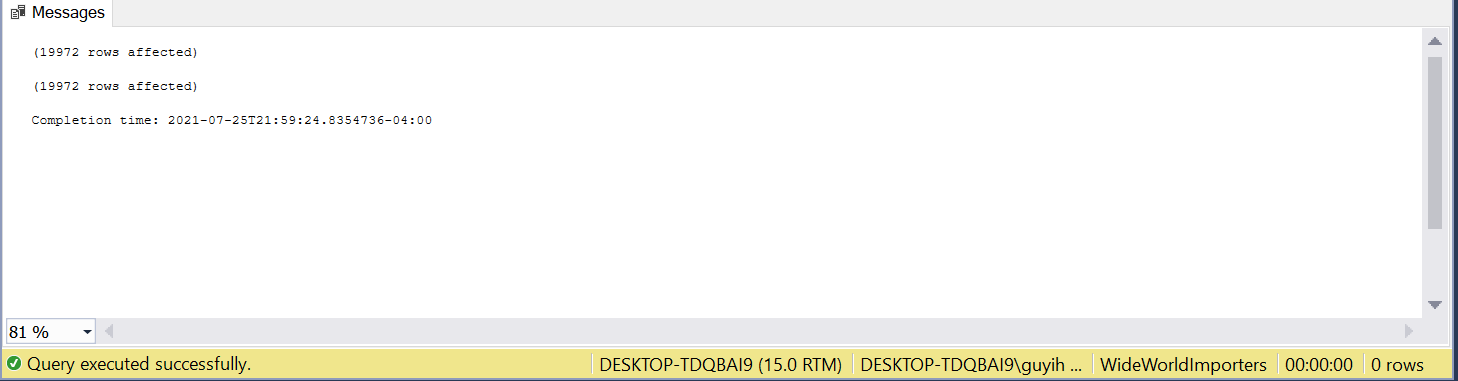
SET @col = SUBSTRING(@col, 0, LEN(@col))

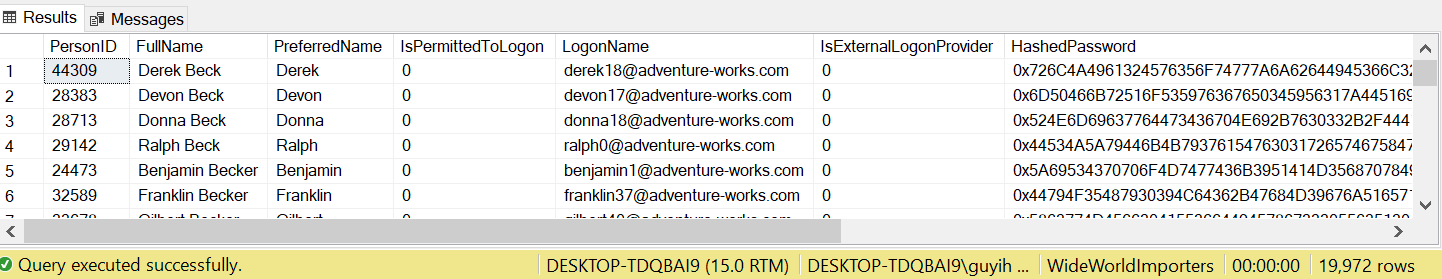
SET @query = 'INSERT INTO Application.People (' + @col +')

SELECT \*

FROM #person'

EXEC(@query)





## This part is to grab all corresponding columns

DECLARE @maxproductid INT;

DECLARE @maxsupplierid INT;

SELECT @maxproductid = MAX(StockItemID)

FROM Warehouse.StockItems

SELECT @maxsupplierid = MAX(SupplierID)

FROM Warehouse.StockItems

SELECT p.ProductID + @maxproductid AS StockItemID, p.[Name] AS StockItemName,

pv.BusinessEntityID + @maxsupplierid AS SupplierID, -1 AS UnitPackageID,

-1 AS OuterPackageID, v.Name AS Brand, p.Size, DaysToManufacture AS LeadTimeDays, -1 AS QuantityPerOuter, 0 AS IsChillerStock, 0 AS TaxRate,

sd.UnitPrice as UnitPrice, -1 AS RecommendedRetailPrice,

p.Weight AS TypicalWeightPerUnit, pr.Comments AS MarketingComments,

1 AS LastEditedBy

INTO #product

FROM AdventureWorks2019.Production.Product as p

JOIN AdventureWorks2019.Purchasing.ProductVendor pv ON p.ProductID = pv.ProductID

JOIN AdventureWorks2019.Purchasing.Vendor v ON pv.BusinessEntityID = v.BusinessEntityID

JOIN AdventureWorks2019.Sales.SalesOrderDetail sd ON sd.ProductID = p.ProductID

JOIN AdventureWorks2019.Production.ProductReview pr ON p.ProductID = pr.ProductID

## This part is to add new supplier id in the supplier table.

DECLARE @maxsupplierid INT;

SELECT @maxsupplierid = MAX(SupplierID)

FROM Purchasing.Suppliers

SELECT v.BusinessEntityID + @maxsupplierid AS SupplierID, v.Name AS SupplierName,

-1 AS SupplierCategoryID, -1 AS AlternateContactPersonID, -1 AS DeliveryCityID,

-1 AS PostalCityID, 0 AS PaymentDays, 'Unknown' AS PhoneNumber,

'Unknown' AS FaxNumber, isnull(v.PurchasingWebServiceURL, 'Unknown') AS WebsiteURL, 'Unknown' AS DeliveryAddressLine1, 'Unknown' AS DeliveryPostalCode,

'Unknown' AS PostalAddressLine1, 'Unknown' AS PostalPostalCode, 1 AS LastEditedBy

INTO #supplier

FROM AdventureWorks2019.Purchasing.Vendor as v

## Finally merge everything into old dataset

INSERT INTO Warehouse.StockItems

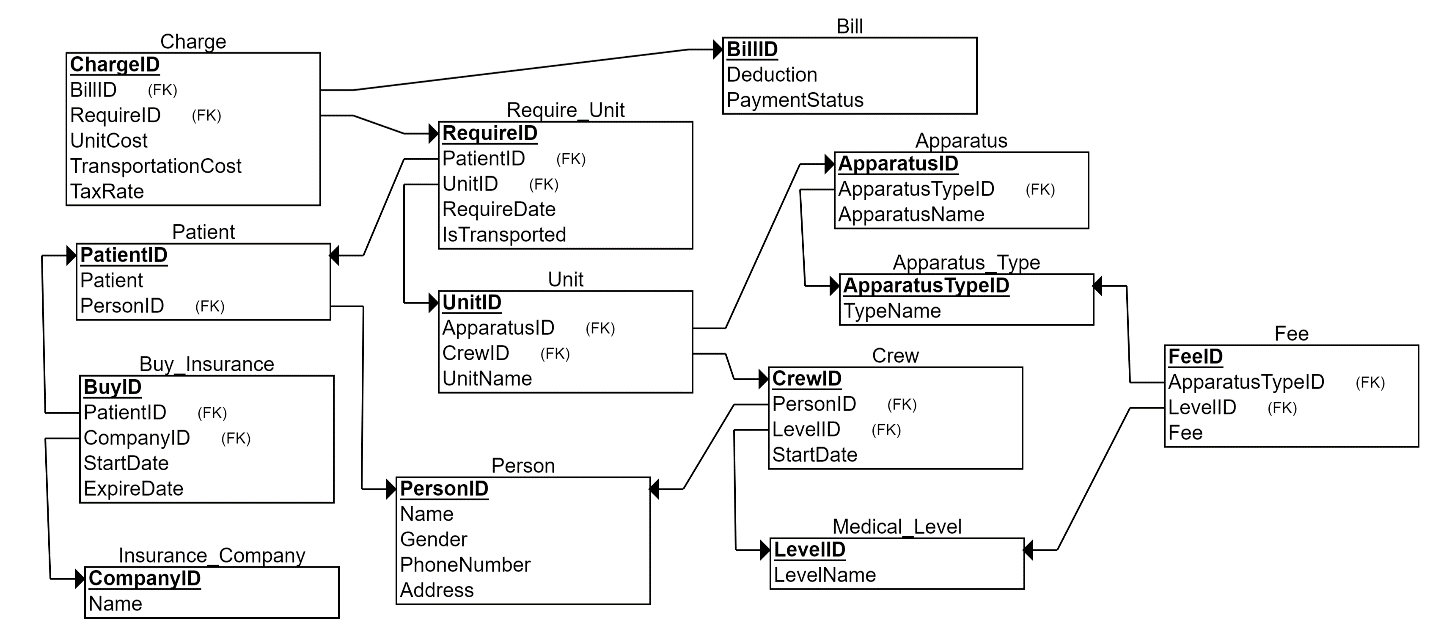
(StockItemID, StockItemName, SupplierID, UnitPackageID, OuterPackageID,

Brand, Size, LeadTimeDays, QuantityPerOuter, IsChillerStock, TaxRate,

UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, MarketingComments, LastEditedBy)

SELECT \* FROM tempdb..#product

32.



34.