1.

Select P.SearchName as PersonName, P.FaxNumber as PersonFax, P.PhoneNumber as PersonNumber, S.PhoneNumber as CompanyNumber, S.FaxNumber as CompanyFax

From WideWorldImporters.Application.People\_Archive as P

JOIN WideWorldImporters.Purchasing.Suppliers\_Archive as S

on P.PersonID = S.PrimaryContactPersonID or P.PersonID = S.AlternateContactPersonID

A picture containing text, newspaper

Description automatically generated

2.

Select DISTINCT(CustomerName) from

Sales.Customers right join

(Select P.PhoneNumber

from Sales.Customers as C

join Application.People as P on C.PrimaryContactPersonID = P.PersonID) as contact

on Sales.Customers.PhoneNumber = contact.PhoneNumber

where CustomerCategoryID != '1'



3.

Select distinct(C.CustomerName)

from Sales.CustomerTransactions as S

join Sales.Customers as C on C.CustomerID = S.CustomerID

where S.TransactionAmount > 0 and S.TransactionDate < '2016-01-01' and S.CustomerID not in

(Select distinct(S1.CustomerID) from Sales.CustomerTransactions as S1

where S1.TransactionAmount > 0 and S1.TransactionDate >= '2016-01-01')

4.

Select I.StockItemName, ABS(stock.TotalAmount) as TotalAmount from

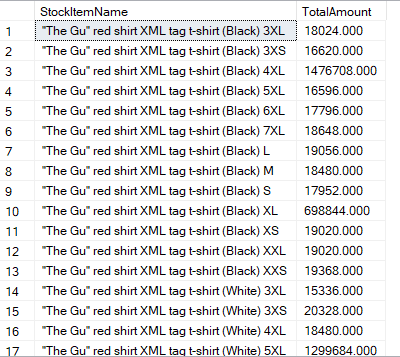
(SELECT StockItemID, sum(Quantity) as TotalAmount, DATEPART(year, TransactionOccurredWhen) as 'Year'

FROM Warehouse.StockItemTransactions as S

group by S.StockItemID, DATEPART(year, S.TransactionOccurredWhen)

having DATEPART(year, S.TransactionOccurredWhen) = 2013) as stock

join Warehouse.StockItems as I on I.StockItemID = stock.StockItemID



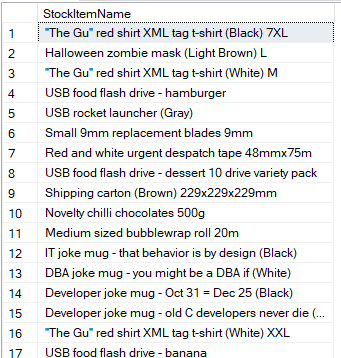
5.

SELECT DISTINCT(S.StockItemName)

FROM Warehouse.StockItems as S

join Sales.OrderLines as O on S.StockItemID = O.StockItemID

where LEN(O.Description) >= 10



6.

Select DISTINCT(SI.StockItemName)

from Sales.OrderLines as OL join Sales.Orders as O on OL.OrderID = O.OrderID

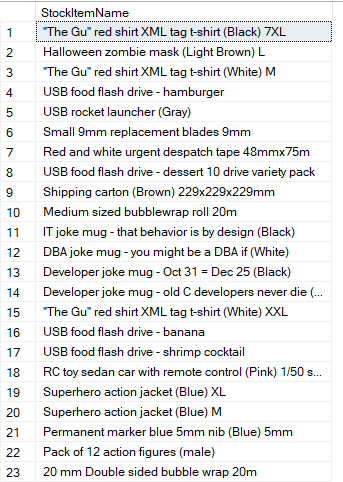
join Sales.Customers as C on C.CustomerID = O.CustomerID

join Application.Cities as City on C.PostalCityID = City.CityID

join Application.StateProvinces as S on S.StateProvinceID = City.StateProvinceID

join Warehouse.StockItems as SI on SI.StockItemID = OL.StockItemID

where DATEPART(year, O.OrderDate) = 2014 and S.StateProvinceName not in ('Alabama', 'Georgia')



7.

Select S.StateProvinceName, AVG(DATEDIFF(day, O.OrderDate, I.ConfirmedDeliveryTime)) as AVG\_Time

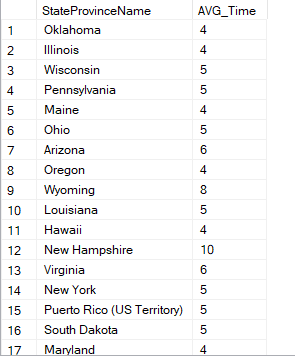
from Sales.Orders as O join Sales.Invoices as I on O.OrderID = I.OrderID

join Sales.Customers as C on C.CustomerID = I.CustomerID

join Application.Cities as City on C.PostalCityID = City.CityID

join Application.StateProvinces as S on S.StateProvinceID = City.StateProvinceID

group by S.StateProvinceName



8.

Select S.StateProvinceName, DATEPART(month, O.OrderDate) as 'Month', AVG(DATEDIFF(day, O.OrderDate, I.ConfirmedDeliveryTime)) as AVG\_Time

from Sales.Orders as O join Sales.Invoices as I on O.OrderID = I.OrderID

join Sales.Customers as C on C.CustomerID = I.CustomerID

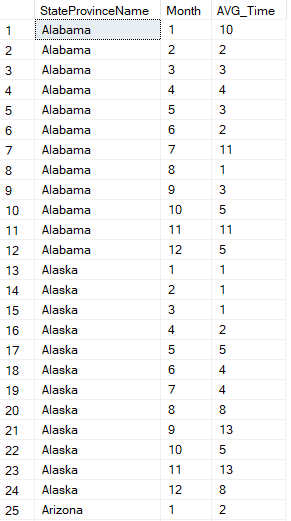
join Application.Cities as City on C.PostalCityID = City.CityID

join Application.StateProvinces as S on S.StateProvinceID = City.StateProvinceID

group by S.StateProvinceName, DATEPART(month, O.OrderDate)

having S.StateProvinceName is not null

order by S.StateProvinceName, DATEPART(month, O.OrderDate)



9.

Select Distinct(SI.StockItemName)

from Purchasing.PurchaseOrders as PO

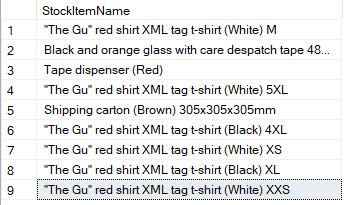
join Purchasing.PurchaseOrderLines as POL on PO.PurchaseOrderID = POL.PurchaseOrderID

join Sales.OrderLines as OL on OL.StockItemID = POL.StockItemID

join Sales.Orders as O on O.OrderID = OL.OrderID

join Warehouse.StockItems as SI on SI.StockItemID = OL.StockItemID

where POL.OrderedOuters-OL.Quantity>0 and DATEPART(year, PO.OrderDate) = 2015 and DATEPART(year, PO.OrderDate) = DATEPART(year, O.OrderDate)



10.

SELECT DISTINCT(C.CustomerName), C.PhoneNumber, P.FullName as ContactPerson

from Sales.Orders as O

join Sales.OrderLines as OL on O.OrderID = OL.OrderID

join Warehouse.StockItems as SI on SI.StockItemID = OL.StockItemID

join Sales.Customers as C on C.CustomerID = O.CustomerID

join Application.People as P on P.PersonID = C.PrimaryContactPersonID

where SI.StockItemName like '%mug%' and DATEPART(year, O.OrderDate) = '2016'

group by C.CustomerName, OL.Quantity, C.PhoneNumber, P.FullName

having sum(OL.Quantity) < 10

Graphical user interface, application

Description automatically generated with medium confidence

11.

DECLARE @CurrentDate date = getDate()

SELECT CityName

from Application.Cities for SYSTEM\_TIME FROM '2015-01-01' to @CurrentDate

Group by CityName

Having count(\*) > 1



12.

Select SI.StockItemName, I.DeliveryInstructions, S.StateProvinceName, City.CityName, CT.CountryName,

C.CustomerName, P.FullName as ContactName, C.PhoneNumber, OL.Quantity

from Sales.Orders as O join Sales.Invoices as I on O.OrderID = I.OrderID

join Sales.OrderLines as OL on OL.OrderID = O.OrderID

join Warehouse.StockItems for SYSTEM\_TIME as of '2014-07-01' as SI on SI.StockItemID = OL.StockItemID

join Sales.Customers as C on C.CustomerID = I.CustomerID

join Application.Cities for SYSTEM\_TIME as of '2014-07-01' as City on C.PostalCityID = City.CityID

join Application.StateProvinces for SYSTEM\_TIME as of '2014-07-01' as S on S.StateProvinceID = City.StateProvinceID

join Application.Countries for SYSTEM\_TIME as of '2014-07-01' as CT on CT.CountryID = S.CountryID

join Application.People for SYSTEM\_TIME as of '2014-07-01' as P on P.PersonID = C.PrimaryContactPersonID

where O.OrderDate = '2014-07-01'



13.

With P as

(SELECT SG.StockGroupID, sum(POL.OrderedOuters) as Purchase

from Purchasing.PurchaseOrderLines as POL

join Warehouse.StockItemStockGroups as SISG on SISG.StockItemID = POL.StockItemID

join Warehouse.StockGroups as SG on SG.StockGroupID = SISG.StockGroupID

group by SG.StockGroupID),

S as

(SELECT SG.StockGroupID, sum(OL.Quantity) as Sale

from Sales.OrderLines as OL

join Warehouse.StockItemStockGroups as SISG on SISG.StockItemID = OL.StockItemID

join Warehouse.StockGroups as SG on SG.StockGroupID = SISG.StockGroupID

group by SG.StockGroupID)

Select P.StockGroupID, P.Purchase, S.Sale, (P.Purchase - S.Sale) as RemainStock

from P join S on P.StockGroupID = S.StockGroupID

order by P.StockGroupID

Table

Description automatically generated

14.

With CC as

(Select DISTINCT(CityName) from Application.Cities as City

join Application.StateProvinces as SP on SP.StateProvinceID = City.StateProvinceID

join Application.Countries as CC on CC.CountryID = SP.CountryID

where CC.CountryName = 'United States'),

R as (Select temp.CityName, ISNULL(temp.StockItemName, 'No Sales') as StockName from

(Select City.CityName, sum(OL.Quantity) as TotalAmount, SI.StockItemName, Rank() over (partition by City.CityName order by sum(OL.Quantity) desc) ranking

from Sales.Orders as O join Sales.Invoices as I on O.OrderID = I.OrderID

join Sales.OrderLines as OL on OL.OrderID = O.OrderID

join Warehouse.StockItems as SI on SI.StockItemID = OL.StockItemID

join Sales.Customers as C on C.CustomerID = I.CustomerID

join Application.Cities as City on C.PostalCityID = City.CityID

join Application.StateProvinces as SP on SP.StateProvinceID = City.StateProvinceID

join Application.Countries as CC on CC.CountryID = SP.CountryID

where CC.CountryName = 'United States' and DATEPART(year, O.OrderDate) = 2016

group by City.CityName, SI.StockItemName) temp

where temp.ranking = 1)

Select CC.CityName, ISNULL(R.StockName, 'No Sales') as StockItem from CC left join R on R.CityName = CC.CityName

Graphical user interface, text

Description automatically generated

15.

Select I.OrderID from Sales.Invoices as I

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

Table

Description automatically generated with medium confidence

16.

Select SI.StockItemName, JSON\_VALUE(SI.CustomFields, '$.CountryOfManufacture') as Country

from Warehouse.StockItems as SI

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

Text

Description automatically generated with low confidence

17.

Select JSON\_VALUE(SI.CustomFields, '$.CountryOfManufacture') as Country, sum(SIT.Quantity) as TotalSale

from Warehouse.StockItems as SI

join Warehouse.StockItemTransactions as SIT on SI.StockItemID = SIT.StockItemID

where DATEPART(year, SIT.TransactionOccurredWhen) = 2015

group by JSON\_VALUE(SI.CustomFields, '$.CountryOfManufacture')

Table

Description automatically generated

18.

CREATE VIEW [Sale of Stock Group] as

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

from

(Select SG.StockGroupName, SISG.StockGroupID, OL.Quantity, DATEPART(year, O.OrderDate) as OrderDate

from Warehouse.StockItems as SI join Warehouse.StockItemStockGroups as SISG on SISG.StockItemID = SI.StockItemID

join Warehouse.StockGroups as SG on SISG.StockGroupID = SG.StockGroupID

join Sales.OrderLines as OL on OL.StockItemID = SI.StockItemID

join Sales.Orders as O on OL.OrderID = O.OrderID

where OL.Quantity > 0) s

PIVOT

(sum(s.Quantity)

for s.OrderDate in ([2013], [2014], [2015], [2016], [2017])) as pvt

order by pvt.StockGroupID

Table

Description automatically generated

19.

CREATE VIEW [Sale of Stock Group] as

Select pvt.OrderDate as [Year], [Novelty Items], [Clothing], [Mugs], [T-Shirts],

isnull([Airline Novelties], 0) as [Airline Novelties], [Computing Novelties], [USB Novelties], [Furry Footwear], [Toys], [Packaging Materials]

from

(Select SG.StockGroupName, OL.Quantity, DATEPART(year, O.OrderDate) as OrderDate

from Warehouse.StockItems as SI join Warehouse.StockItemStockGroups as SISG on SISG.StockItemID = SI.StockItemID

join Warehouse.StockGroups as SG on SISG.StockGroupID = SG.StockGroupID

join Sales.OrderLines as OL on OL.StockItemID = SI.StockItemID

join Sales.Orders as O on OL.OrderID = O.OrderID

where OL.Quantity > 0) s

PIVOT

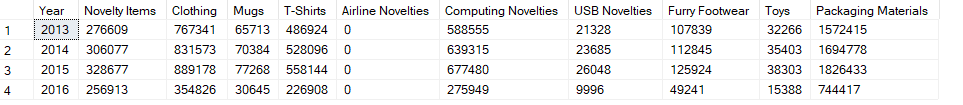
(sum(s.Quantity)

for s.StockGroupName in ([Novelty Items], [Clothing], [Mugs], [T-Shirts],

[Airline Novelties], [Computing Novelties], [USB Novelties],

[Furry Footwear], [Toys], [Packaging Materials])) as pvt

Order by [Year]



20.

CREATE FUNCTION Sales.AttachOrder (

@id INT

)

RETURNS DECIMAL(18,2) AS

Begin

RETURN (Select sum(OL.Quantity\*OL.UnitPrice) as Total

from Sales.OrderLines as OL

here OL.OrderID = @id)

END

A picture containing table

Description automatically generated

21.

CREATE SCHEMA ods

CREATE TABLE ods.Orders

(

OrderId int not null,

OrderDate datetime2 not null,

OrderTotal decimal(18,2) null,

CustomerID int not null

)

CREATE PROCEDURE Sales.OrderCaculate

@Date datetime2

AS

SET NOCOUNT OFF;

If (@Date in (select DISTINCT(Orders.OrderDate) from ods.Orders))

BEGIN

RAISERROR ('Date already in the dataset.',

16,

1)

END

ELSE

BEGIN

INSERT INTO ods.Orders

Select O.OrderID, O.OrderDate, sum(OL.Quantity\*OL.UnitPrice) AS TotalOrder, O.CustomerID

from Sales.Orders as O

join Sales.OrderLines as OL on OL.OrderID = O.OrderID

group by O.OrderID, O.OrderDate, O.CustomerID

having O.OrderDate = @Date

END

Table

Description automatically generated

22.

CREATE TABLE ods.StockItem

(

StockItemID int PRIMARY KEY not null,

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) not null,

TypicalWeightPerUnit decimal(18,3) not null,

MarketingComments nvarchar(max) null,

InternalComments nvarchar(max) null,

CountryOfManufacture nvarchar(20) null,

[Range] nvarchar(100) null,

ShelfLife nvarchar(100) null

)

INSERT INTO StockItem

Select SI.StockItemID, SI.StockItemName, SI.SupplierID,

SI.ColorID, SI.UnitPackageID, SI.OuterPackageID, SI.Brand ,SI.Size,

SI.LeadTimeDays, SI.QuantityPerOuter, SI.IsChillerStock, SI.Barcode,

SI.TaxRate, SI.UnitPrice, SI.RecommendedRetailPrice, SI.TypicalWeightPerUnit, SI.MarketingComments, SI.InternalComments, JSON\_VALUE(SI.CustomFields, '$.CountryOfManufacture') as CountryOfManufacture,

JSON\_VALUE(SI.CustomFields, '$.Range') as [Range],

JSON\_VALUE(SI.CustomFields, '$.ShelfLife') as ShelfLife

from Warehouse.StockItems as SI

A picture containing text, indoor, shelf, decorated

Description automatically generated

23.

CREATE PROCEDURE Sales.OrderCaculate2

@Date datetime2

AS

SET NOCOUNT OFF;

BEGIN TRY

BEGIN TRANSACTION

DELETE FROM dbo.Orders

select \* from dbo.Orders where dbo.Orders.OrderDate < @Date

INSERT INTO dbo.Orders

Select O.OrderID, O.OrderDate, sum(OL.Quantity\*OL.UnitPrice) AS TotalOrder, O.CustomerID

from Sales.Orders as O

join Sales.OrderLines as OL on OL.OrderID = O.OrderID

group by O.OrderID, O.OrderDate, O.CustomerID

having O.OrderDate BETWEEN DATEADD(day, 1, @Date) AND DATEADD(day, 7, @Date)

COMMIT TRANSACTION

END TRY

BEGIN CATCH

END CATCH

Table

Description automatically generated

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 a as

(Select \*

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

WITH (

StockItemID int,

StockItemName nvarchar(100),

SupplierID int '$.Supplier',

ColorID int,

UnitPackageID int '$.UnitPackageId',

OuterPackageID int '$.OuterPackageId',

Brand nvarchar(50),

Size nvarchar(20),

LeadTimeDays int,

QuantityPerOuter int,

IsChillerStock bit,

Barcode nvarchar(50),

TaxRate decimal(18,3),

UnitPrice decimal(18,2),

RecommendedRetailPrice decimal(18,2),

TypicalWeightPerUnit decimal(18,3),

MarketingComments nvarchar(max),

InternalComments nvarchar(max),

Photo varbinary(max),

CustomFields nvarchar(max),

Tags nvarchar(max),

SearchDetails nvarchar(max),

LastEditedBy int,

ValidFrom datetime2(7),

ValidTo datetime2(7)))

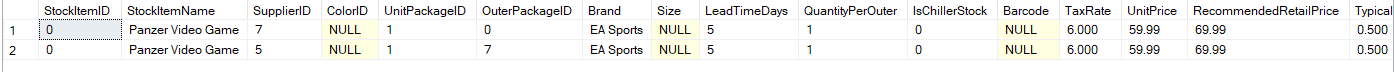
INSERT INTO Warehouse.StockItems

select isnull(StockItemID, 0) as StockItemID, StockItemName, SupplierID, ColorID,

isnull(UnitPackageID, 0) as UnitPackageID, isnull(OuterPackageID, 0) as OuterPackageID, Brand, Size, isnull(LeadTimeDays, 0) as LeadTimeDays, isnull(QuantityPerOuter, 0) as QuantityPerOuter,

isnull(IsChillerStock, 0) as IsChillerStock, Barcode, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, MarketingComments, InternalComments, Photo, CustomFields, Tags, isnull(SearchDetails, 0) as SearchDetails, isnull(LastEditedBy, 0) as LastEditedBy,

isnull(ValidFrom, GETDATE()) as ValidFrom, isnull(ValidTo, '9999-12-30 23:59:59') as ValidTo from a



with b as

(Select \*

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

WITH (

PurchaseOrderID int,

SupplierID int '$.Supplier',

OrderDate date,

DeliveryMethodID int,

ContactPersonID int,

ExpectedDeliveryDate date '$.ExpectedDeliveryDate',

SupplierReference nvarchar(20),

IsOrderFinalized bit,

Comments nvarchar(max),

InternalComments nvarchar(max),

LastEditedBy int,

LastEditedWhen datetime2))

INSERT INTO Purchaing.PurchaseOrderLines

Select isnull(PurchaseOrderID, 0) as PurchaseOrderID, SupplierID, isnull(OrderDate, 0) as OrderDate,

isnull(DeliveryMethodID, 0) as DeliveryMethodID, isnull(ContactPersonID, 0) as ContactPersonID,

ExpectedDeliveryDate, SupplierReference, isnull(IsOrderFinalized, 0) as IsOrderFinalized,

Comments, InternalComments, isnull(LastEditedBy, 0) as LastEditedBy,

isnull(LastEditedWhen, GETDATE()) as LastEditedWhen from b

with c as

(Select \*

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

WITH (

PurchaseOrderLineID int,

PurchaseOrderID int,

StockItemID int,

OrderedOuters int,

[Description] nvarchar(100),

ReceivedOuters int,

PackageTypeID int,

ExpectedUnitPricePerOuter decimal(18,2),

LastReceiptDate date,

IsOrderLineFinalized bit,

LastEditedBy int,

LastEditedWhen datetime2))

Select isnull(PurchaseOrderLineID, 0) as PurchaseOrderLineID, isnull(PurchaseOrderID, 0) as PurchaseOrderID,

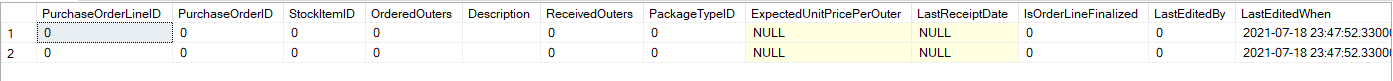
isnull(StockItemID, 0) as StockItemID, isnull(OrderedOuters, 0) as OrderedOuters,

isnull([Description], '') as [Description], isnull(ReceivedOuters, 0) as ReceivedOuters,

isnull(PackageTypeID, 0) as PackageTypeID, ExpectedUnitPricePerOuter, LastReceiptDate,

isnull(IsOrderLineFinalized, 0) as IsOrderLineFinalized, isnull(LastEditedBy, 0) as LastEditedBy,

isnull(LastEditedWhen, GETDATE()) as LastEditedWhen from c



25.

Select OrderDate as [Year],

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

[Clothing] AS 'StockGroup.Clothing',

[Mugs] AS 'StockGroup.Mugs',

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

isnull([Airline Novelties], 0) 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 dbo.[Sale of Stock Group] as s

FOR JSON PATH

A picture containing chart

Description automatically generated

26.

SELECT [OrderDate] 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 dbo.[Sale of Stock Group]

FOR XML PATH('StockItems')

Timeline

Description automatically generated

27.

CREATE TABLE ods.ConfirmedDeliveryJson

(

id int not null,

[date] datetime2 null,

value int null

)

CREATE FUNCTION Sales.Invoices.JsonTransform (

@date datetime2

)

RETURNS TABLE AS

RETURN

Select I.\*, IL.\* from Sales.Invoices as I

join Sales.InvoiceLines as IL on I.InvoiceID = IL.InvoiceID

where @date= I.ConfirmedDeliveryTime

FOR JSON AUTO

CREATE PROCEDURE Sales.DeliverySummary

@Date datetime2

AS

SET NOCOUNT OFF;

BEGIN TRY

BEGIN TRANSACTION

INSERT INTO ods.ConfirmedDeliveryJson

Select I.InvoiceID as 'id', I.ConfirmedDeliveryTime as 'date',

Sales.Invoices.JsonTransform(I.ConfirmedDeliveryTime) as 'value'

from Sales.Invoices as I

join Sales.InvoiceLines as IL on I.InvoiceID = IL.InvoiceID

where @Date = I.ConfirmedDeliveryTime

COMMIT TRANSACTION

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION

END CATCH

28.

As defined, a transaction is a sequence of operations performed as a single logical unit of work. It is similar as a systematic machine with strict rules to perform the tasks user required. The rules can be simplified as four properties as atomicity, consistency, isolation, and durability (ACID). It means that transaction must be stable, atomic, and accessed in isolation. There are also many types of different transactions based on user needs. Autocommit transaction is the default transaction management mode of the SQL Server Database Engine, meaning that whenever you run SQL statement, this kind of transaction will be triggered. If there are no errors, statements can be committed, and the system will rollback completely if not so. Another kind of transaction is implicit transaction, and it means that transactions in the instance will be prosecuted one by one without specifying anything. A continuous chain of transactions can be generated under such situation.

As defined, lock is a mechanism used by the SQL Server Database Engine to synchronize access by multiple users to the same piece of data at the same time. Different users could access the same file simultaneously and if they try to change and update the table at the same time, there would be serious conflict happening for the table. For this reason, locks are introduced to avoid such confusion and usually will restrict the permission of using the data we care about. Locks are managed internally by a part of the SQL Server Database Engine called the lock manager and the lock manager will judge which kind of lock to be used under different requirements. There are also many kinds of lock, namely shared lock, update lock and exclusive lock.

Finally, as defined, transactions specify an isolation level that defines the degree to which one transaction must be isolated from resource or data modifications made by other transactions. The rule of naming different kind of isolation level is related with various side effects of concurrency. Basically, the highest level is serializable, meaning that every transaction is isolated from others. Such kind of isolation level can eliminate all side effects of concurrency, but the overhead is big, and the efficiency is low. On the other hand, read uncommitted is the lowest level of isolation level with lots of flexible but at the expense of potential data lost. Read committed and Repeatable read are two intermediate level and they are commonly used in the current industry because users need some flexibility to ensure the efficiency but also need the restriction to make sure all updates are not messed up.

29.

The first thing I will do is to email boss that I will investigate the issue immediately. Then I will try to simulate the error on the website first and see how slow the website it was. The potential reason could be various. It could be the issue of server itself or it could be the query to grab data for importing to the website runs too long. Thus, after checking the details, I will stop the service first and have a basic guess. If I believe it is the issue from the long-time running query, I will use extended events to set parameters like sql\_statement and sql\_batch to check the length of execution time. Another issue I want to check is whether we have a dead lock for the statements so we can also use extended events to check it. After I know the exact issue, I will estimate the time of fixing the errors and give the schedule to my boss. For the short term, I can think about a hot fix if the user wants the error fix as soon as possible but for the long term, I have to think the source of error and design a stable solution for that to avoid happening similar issue in future. Probably some queries are not efficient enough and needed to refactor. At the same time, I will use execution plan to improve the performance of all queries and make sure the running time of all queries are within the reasonable range. I will also schedule a daily-basis checking for the dataset by using DMV, which returns the server-state information for the purpose of diagnose the health of the system.

30.

Deadlock:

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Long-run Query

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

31.

For merging Logon and Person information into World Wide Importers 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.

Graphical user interface

Description automatically generated with medium confidence

## This part is to merge data into person table

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)

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.

## 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.

The dataset our group designed is as follows. We have 12 tables in total with necessary primary keys and foreign keys assigned. In the diagram of structure, the relationships between tables are also shown. Mainly, the system consists of three parts with patient, charges and tools used for the emergency.

Diagram

Description automatically generated

34.