SQL Assignments
ChengzhuangZheng

SQL related assignments will be on Wide World Importers Database if not otherwise introduced.

1. **List of Persons' full name, all their fax and phone numbers, as well as the phone number and fax of the company they are working for (if any).**

   SELECT p.FullName, a.FaxNumber, a.PhoneNumber, a.CompanyPhoneNumber, a.CompanyFaxNumber
   FROM
   (SELECT p.FullName, p.FaxNumber, p.PhoneNumber, s.PhoneNumber as CompanyPhoneNumber, s.FaxNumber as CompanyFaxNumber
   FROM Application.People p
   JOIN Purchasing.Suppliers s
   ON s.PrimaryContactPersonID = p.PersonID
   UNION
   SELECT p.FullName, p.FaxNumber, p.PhoneNumber, c.PhoneNumber as CompanyPhoneNumber, c.FaxNumber as CompanyFaxNumber
   FROM Sales.Customers c
   JOIN Application.People p
   ON c.PrimaryContactPersonID = p.PersonID) a
   RIGHT JOIN Application.People p
   ON a.FullName = p.FullName

2. **If the customer's primary contact person has the same phone number as the customer's phone number, list the customer companies.**

   SELECT CustomerName
   FROM [WideWorldImporters].[Sales].[Customers] c
   JOIN [WideWorldImporters].[Application].[People] p
   ON c.PrimaryContactPersonID = p.PersonId
   WHERE c.PhoneNumber = p.PhoneNumber;

3. **List of customers to whom we made a sale prior to 2016 but no sale since 2016-01-01.**

```
select distinct CustomerID
from [WideWorldImporters].[Sales].[CustomerTransactions]
where YEAR(TransactionDate) <'2016'

intersect

select CustomerID
from [WideWorldImporters].[Sales].[CustomerTransactions]
group by CustomerID
having max(TransactionDate) <'2016-01-01'
```

4. **List of Stock Items and total quantity for each stock item in Purchase Orders in Year 2013.**

```
SELECT StockItemID, sum(OrderedOuters) AS [TOTAL Quantity]
FROM [WideWorldImporters].[Purchasing].[PurchaseOrderLines] pol
JOIN (SELECT PurchaseOrderID, OrderDate
        FROM[WideWorldImporters].[Purchasing].[PurchaseOrders]
        WHERE YEAR(OrderDate) = '2013') po
ON pol.PurchaseOrderID = po.PurchaseOrderID
GROUP BY StockItemID;
```

5. **List of stock items that have at least 10 characters in description.**

```
select distinct StockItemID
from
[WideWorldImporters].[Purchasing].[PurchaseOrderLines]
where len( Description)>= 10;
```

6. **List of stock items that are not sold to the state of Alabama and Georgia in 2014.**

```
SELECT DISTINCT(StockItemID)
FROM Sales.OrderLines
WHERE StockItemID NOT IN (SELECT StockItemID
                          FROM Sales.OrderLines ol
                          JOIN Sales.Orders o
                          ON ol.ORDERID = o.OrderID
                          JOIN Sales.Customers c
                          ON o.CustomerID = c.CustomerID
```

JOIN Application.Cities ci
ON c.DeliveryCityID = ci.CityID
JOIN Application.StateProvinces sp
ON ci.StateProvinceID = sp.StateProvinceID
WHERE StateProvinceName IN ('Alabama', 'Georgia')
AND YEAR(OrderDate) = '2014');

7. **List of States and Avg dates for processing (confirmed delivery date – order date).**

```
select city.StateProvinceID as stateID,avg( DATEDIFF(day,o.OrderDate
,CONVERT(DATE,
i.ConfirmedDeliveryTime))) as averageprocessing
        from [WideWorldImporters].[Sales].[Orders] o
        join [WideWorldImporters].[Sales].[Invoices] i
        on i.OrderID= o.OrderID
        join [WideWorldImporters].[Sales].[Customers] c
        on c.CustomerID = o.CustomerID
        join [WideWorldImporters].[Application].[Cities] city
        on city.CityID = c.DeliveryCityID
        group by city.StateProvinceID
```

8. **List of States and Avg dates for processing (confirmed delivery date – order date) by month.**

```
select city.StateProvinceID as stateID , o.month , avg( DATEDIFF(day , o.OrderDate
,CONVERT(DATE, i.ConfirmedDeliveryTime))) as averageprocessing
from(select *, month(OrderDate) as month from
[WideWorldImporters].[Sales].[Orders] )o
join [WideWorldImporters].[Sales].[Invoices] i
on i.OrderID= o.OrderID
join [WideWorldImporters].[Sales].[Customers] c
on c.CustomerID = o.CustomerID
join [WideWorldImporters].[Application].[Cities] city
on city.CityID = c.DeliveryCityID
group by city.StateProvinceID, o.month
order by   city.StateProvinceID,o.month
```

9. **List of StockItems that the company purchased more than sold in the year of 2015.**

```
select StockItemID,sum(Quantity) as Quantitysum
from [WideWorldImporters].[Warehouse].[StockItemTransactions]
where YEAR(TransactionOccurredWhen) = 2015
group by StockItemID
having(sum(Quantity)>0)
```

10. **List of Customers and their phone number, together with the primary contact person's name, to whom we did not sell more than 10 mugs (search by name) in the year 2016.**

```
select cu.CustomerID ,cu.PhoneNumber,peo.FullName
from  [WideWorldImporters].[Sales].[Customers] cu
join [WideWorldImporters].[Application].[People] peo
on peo.PersonID = cu.PrimaryContactPersonID
where cu.CustomerID in

(select c.CustomerID
from  [WideWorldImporters].[Sales].[Customers] c
join  [WideWorldImporters].[Sales].[Orders] o
on c.CustomerID = o.CustomerID
join [WideWorldImporters].[Sales].[OrderLines] ol
on ol.OrderID = o.OrderID
join Warehouse.StockItems si
ON ol.StockItemID = si.StockItemID
```

```
where Year(o.OrderDate) ='2016'
AND
si.StockItemName like '%mug%'
group by c.CustomerID
having sum(ol.Quantity)<=10
)
```

11. List all the cities that were updated after 2015-01-01.

```
select  distinct CityName
from [WideWorldImporters].[Application].[Cities]
where ValidFrom >'2015-01-01'
```

12. List all the Order Detail (Stock Item name, delivery address, delivery state, city, country, customer name, customer contact person name, customer phone, quantity) for the date of 2014-07-01. Info should be relevant to that date.

```
select st.StockItemName , concat(c.DeliveryAddressLine1,'
,',c.DeliveryAddressLine2,' ',',c.DeliveryPostalCode) as 'delivery address' ,
statee.StateProvinceName , ci.CityName , coun.CountryName ,c.CustomerName
,pe.FullName , c.PhoneNumber , ol.Quantity
from  [WideWorldImporters].[Sales].[Customers] c
join  ( select * from [WideWorldImporters].[Sales].[Orders] where
OrderDate='2014-07-01' ) o
on c.CustomerID = o.CustomerID
join [WideWorldImporters].[Sales].[OrderLines] ol
```

```
on ol.OrderID = o.OrderID
left outer join [WideWorldImporters].[Warehouse].[StockItems] st
on st.StockItemID = ol.StockItemID
left outer join [WideWorldImporters].[Application].[Cities] ci
on c.DeliveryCityID =ci.CityID
left outer join  [WideWorldImporters].[Application].[StateProvinces] statee
on ci.StateProvinceID=statee.StateProvinceID
left outer join [WideWorldImporters].[Application].[Countries] coun
on coun.CountryID = statee.CountryID
join [WideWorldImporters].[Application].[People] pe
on pe.PersonID =c.PrimaryContactPersonID
```

13. **List of stock item groups and total quantity purchased, total quantity sold, and the remaining stock quantity (quantity purchased – quantity sold)**

```
select  ssg.StockGroupID , sum(case when Quantity>0 then Quantity else 0 end)
as purchased , sum(case when Quantity<0 then Quantity*(-1) else 0 end) as sold,
sum(st.Quantity) as remaining
FROM
[WideWorldImporters].[Warehouse].[StockItemTransactions] st
join

[WideWorldImporters].[Warehouse].[StockItemStockGroups] ssg
on ssg.StockItemID = st.StockItemID

group by ssg.StockGroupID
```

14. **List of Cities in the US and the stock item that the city got the most deliveries in 2016. If the city did not purchase any stock items in 2016, print "No Sales".**

```
with cte1 as (
select cit.CityName
From
```

```
[WideWorldImporters].[Application].[Cities] cit

join  [WideWorldImporters].[Application].[StateProvinces] statee
on statee.StateProvinceID = cit.StateProvinceID

join (select * from [WideWorldImporters].[Application].[Countries] where
CountryName = 'United States') count
on count.CountryID = statee.CountryID

),


cte2 as (


select CityName, StockItemID
FROM
(
select CityName, StockItemID , rank() over(partition by CityName order by
numdeli desc) as ranking
FROM

(select cit.CityName, orl.StockItemID , count(inv.ConfirmedDeliveryTime) as
numdeli
FROM
[WideWorldImporters].[Application].[Cities] cit

join  [WideWorldImporters].[Application].[StateProvinces] statee
on statee.StateProvinceID = cit.StateProvinceID

join (select * from [WideWorldImporters].[Application].[Countries] where
CountryName = 'United States') count
on count.CountryID = statee.CountryID


join
 [WideWorldImporters].[Sales].[Customers] c
on cit.CityID =  c.DeliveryCityID

join (select * from [WideWorldImporters].[Sales].[Orders]) o
on o.CustomerID = c.CustomerID
```

```
join (SELECT * FROM [WideWorldImporters].[Sales].[Invoices] where
YEAR(ConfirmedDeliveryTime) = '2016') inv
on o.CustomerID = inv.CustomerID


join [WideWorldImporters].[Sales].[OrderLines] orl
on orl.OrderID=  inv.OrderID
group by  cit.CityName, orl.StockItemID
) tempt
) tempt2

where ranking  = 1)




select cte1.CityName,
        case when( cte2.StockItemID  is not null) then cte2.StockItemID
                                    else  'Not Sales'
                                    end as moststockitem
from
cte1
left outer join cte2
on cte2.CityName = cte1.CityName
```

15. List any orders that had more than one delivery attempt (located in invoice table).

```
 select OrderID
FROM (select distinct OrderID, rank() over(order by deliverattempt desc)  as
ranking
FROM
(SELECT OrderID , len(JSON_VALUE(inv.ReturnedDeliveryData,
'$.Events[1].Event')) as deliverattempt
FROM
[WideWorldImporters].[Sales].[Invoices] inv
where JSON_VALUE(inv.ReturnedDeliveryData, '$.Events[1].Event') is not null)
tempt
```

```
) temptt

where ranking >1
```

16. List all stock items that are manufactured in China. (Country of Manufacture)

```
select StockItemID
FROM(select
StockItemID,JSON_VALUE(st.CustomFields,'$.CountryOfManufacture') as count
FROM  [WideWorldImporters].[Warehouse].[StockItems] st
) tempt
where count='China'
```

17. Total quantity of stock items sold in 2015, group by country of manufacturing.

```
select count, sum(orl.Quantity)          as totalsold
FROM
[WideWorldImporters].[Sales].[OrderLines] orl
join

(select StockItemID,JSON_VALUE(st.CustomFields,'$.CountryOfManufacture') as
count
FROM  [WideWorldImporters].[Warehouse].[StockItems] st
) tempt
on tempt.StockItemID = orl.StockItemID
join [WideWorldImporters].[Sales].[Orders] orr
on orr.OrderID = orl.OrderID
where Year(orr.OrderDate) = '2015'
group by count
```

18. Create a view that shows the total quantity of stock items of each stock group sold (in orders) by year 2013-2017. [Stock Group Name, 2013, 2014, 2015, 2016, 2017]

```
CREATE VIEW view1 as
```

```
with cte as (select StockGroupName,  Year(OrderDate)  as orderyear  , Quantity
FROM (select * from [WideWorldImporters].[Sales].[Orders]
where Year(OrderDate) in (2013, 2014, 2015 , 2016, 2017) ) o
join [WideWorldImporters].[Sales].[OrderLines] orl
on orl.OrderID = o.OrderID
join [WideWorldImporters].[Warehouse].[StockItemStockGroups] ssg
on orl.StockItemID = ssg.StockItemID
JOIN Warehouse.StockGroups sgs
ON ssg.StockGroupID = sgs.StockGroupID
)


select StockGroupName, [2013], [2014], [2015] , [2016], [2017]
from cte
pivot
(sum(Quantity) for orderyear in ([2013], [2014], [2015] , [2016], [2017]
)) pvt
```

19. **Create a view that shows the total quantity of stock items of each stock group sold (in orders) by year 2013-2017. [Year, Stock Group Name1, Stock Group Name2, Stock Group Name3, … , Stock Group Name10]**

```
DECLARE @col as nvarchar(max);
DECLARE @query as nvarchar(max);


SELECT @col = COALESCE(@col + ', ', '') + QUOTENAME([StockGroupName])
FROM (SELECT DISTINCT StockGroupName FROM Warehouse.StockGroups) a



set @query = 'select orderyear,' + @col + '
FROM (select StockGroupName,  Year(OrderDate)  as orderyear  , Quantity
FROM (select * from [WideWorldImporters].[Sales].[Orders]
where Year(OrderDate) in (2013, 2014, 2015 , 2016, 2017) ) o
join [WideWorldImporters].[Sales].[OrderLines] orl
on orl.OrderID = o.OrderID
join [WideWorldImporters].[Warehouse].[StockItemStockGroups] ssg
on orl.StockItemID = ssg.StockItemID
JOIN Warehouse.StockGroups sgs
```

```
ON ssg.StockGroupID = sgs.StockGroupID
) tempt
PIVOT(
sum(Quantity) for StockGroupName in (' + @col + ')
) pvt'

exec(@query)


DECLARE @tabb AS nvarchar(max)

select @tabb = 'create view viewe as '+ @query

EXEC(@tabb)
```

20. **Create a function, input: order id; return: total of that order. List invoices and use that function to attach the order total to the other fields of invoices.**

```
create function funcc(@orderidfind INT)
RETURNS TABLE
AS
RETURN
(select inv.*, orl.Quantity * orl.UnitPrice
as OrderTotal
from
[WideWorldImporters].[Sales].[Invoices] inv
join [WideWorldImporters].[Sales].[OrderLines] orl
on inv.OrderID = orl.OrderID
WHERE inv.OrderId = @orderidfind
)
GO

select * from funcc(202)
```

21. **Create a new table called ods.Orders. Create a stored procedure, with proper error handling and transactions, that input is a date; when executed, it would**

find orders of that day, calculate order total, and save the information (order id, order date, order total, customer id) into the new table. If a given date is already existing in the new table, throw an error and roll back. Execute the stored procedure 5 times using different dates.

```
CREATE TABLE ods.Orders (
OrderID INT,
OrderDate DATE ,
OrderTotal FLOAT,
CustomerID  INT
)
GO

CREATE PROCEDURE process_searchdate
        @datee DATE
AS
BEGIN TRY
        BEGIN TRANSACTION


        INSERT INTO ods.Orders
        select orl.OrderID ,  o.OrderDate  ,orl.Quantity* orl.UnitPrice as
        OrderTotal , o.CustomerID
        FROM [WideWorldImporters].[Sales].[OrderLines] orl
        JOIN [WideWorldImporters].[Sales].[Orders] o
        ON o.OrderID = orl.OrderID
        where o.OrderDate = @datee


        COMMIT TRANSACTION
END TRY

BEGIN CATCH
        if  EXISTS (select orl.OrderID ,  o.OrderDate  ,orl.Quantity* orl.UnitPrice as
        OrderTotal , o.CustomerID
        FROM [WideWorldImporters].[Sales].[OrderLines] orl
        JOIN [WideWorldImporters].[Sales].[Orders] o
        ON o.OrderID = orl.OrderID
        where o.OrderDate = @datee )
```

```
                    PRINT 'Duplicate Date';
                    ROLLBACK TRANSACTION;
END CATCH

GO

EXEC process_searchdate @datee = '2013-01-01';
EXEC process_searchdate @datee = '2013-01-02;
EXEC process_searchdate @datee = '2013-01-03';
EXEC process_searchdate @datee = '2013-01-04';
EXEC process_searchdate @datee = '2013-01-05';
```

22. Create a new table called ods.StockItem. It has following columns: [StockItemID], [StockItemName] ,[SupplierID] ,[ColorID] ,[UnitPackageID] ,[OuterPackageID] ,[Brand] ,[Size] ,[LeadTimeDays] ,[QuantityPerOuter] ,[IsChillerStock] ,[Barcode] ,[TaxRate] ,[UnitPrice],[RecommendedRetailPrice] ,[TypicalWeightPerUnit] ,[MarketingComments] ,[InternalComments], [CountryOfManufacture], [Range], [Shelflife]. Migrate all the data in the original stock item table.

```
CREATE TABLE ods.StockItem (
        StockItemID INT NOT NULL 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(100) NULL ,
```

```
Range NVARCHAR(100) NULL,
Shelflife DATE NULL
);



INSERT INTO ods.StockItem
select  [StockItemID], [StockItemName] ,[SupplierID] ,[ColorID] ,
[UnitPackageID] ,[OuterPackageID] ,[Brand] ,[Size] ,[LeadTimeDays]
,[QuantityPerOuter] ,[IsChillerStock] ,[Barcode] ,[TaxRate]
,[UnitPrice],[RecommendedRetailPrice] ,[TypicalWeightPerUnit]
,[MarketingComments]  ,[InternalComments],
JSON_VALUE(si.CustomFields,'$.CountryOfManufacture'),JSON_VALUE(si.Custom
Fields,'$.Range'),NULL
FROM
        [WideWorldImporters].[Warehouse].[StockItems]  si
```

23. Rewrite your stored procedure in (21). Now with a given date, it should wipe out all the order data prior to the input date and load the order data that was placed in the next 7 days following the input date.

```
CREATE TABLE ods.Orders (
OrderID INT,
OrderDate DATE ,
OrderTotal FLOAT,
CustomerID  INT
)

GO

CREATE PROCEDURE process_searchdate
        @datee DATE

AS
BEGIN TRY
        BEGIN TRANSACTION
        DELETE FROM ods.Orders
        WHERE OrderDate <@datee

        INSERT INTO ods.Orders
```

```
        select orl.OrderID ,  o.OrderDate  ,orl.Quantity* orl.UnitPrice as
        OrderTotal , o.CustomerID
        FROM [WideWorldImporters].[Sales].[OrderLines] orl
        JOIN [WideWorldImporters].[Sales].[Orders] o
        ON o.OrderID = orl.OrderID
          WHERE OrderDate < DATEADD(DD,7,@datee) AND OrderDate >=@datee



        COMMIT TRANSACTION
    END TRY

    BEGIN CATCH
        if  EXISTS (select orl.OrderID ,  o.OrderDate  ,orl.Quantity* orl.UnitPrice as
        OrderTotal , o.CustomerID
        FROM [WideWorldImporters].[Sales].[OrderLines] orl
        JOIN [WideWorldImporters].[Sales].[Orders] o
        ON o.OrderID = orl.OrderID
        where o.OrderDate = @datee );

        PRINT 'Duplicate Date';
        ROLLBACK TRANSACTION;
    END CATCH

    GO

    EXEC process_searchdate @datee = '2015-03-01';
```

24. Consider the JSON file:


```
{
  "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"
    }
  ]
}
```


Looks like that it is our missed purchase orders. Migrate these data into Stock Item, Purchase Order and Purchase Order Lines tables. Of course, save the script.

```
DECLARE @jsonda nvarchar(max);

set @jsonda =  '{
  "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"
      }
   ]
}';



INSERT INTO  [WideWorldImporters].[Warehouse].[StockItems]
SELECT *
FROM OPENJSON(@jsonda)
WITH(
StockItemID int '999',
StockItemName nvarchar(100)  '$.PurchaseOrders.StockItemName',
SupplierID  int '$.PurchaseOrders.Supplier' ,
UnitPackageId   int      '$.PurchaseOrders.UnitPackageId' ,
OuterPackageId     int    '$.PurchaseOrders.OuterPackageId[0]' ,
Brand    nvarchar(50)    '$.PurchaseOrders.Brand' ,
LeadTimeDays   int    '$.PurchaseOrders.LeadTimeDays',
QuantityPerOuter int '$.PurchaseOrders.QuantityPerOuter' ,
IsChillerStock bit  '0',
 TaxRate     decimal(18,3)   '$.PurchaseOrders.TaxRate',
  UnitPrice    decimal(18,2)   '$.PurchaseOrders.UnitPrice',
   RecommendedRetailPrice decimal(18,2)
'$.PurchaseOrders.RecommendedRetailPrice',
    TypicalWeightPerUnit   decimal(18,3)
'$.PurchaseOrders.TypicalWeightPerUnit',
   [CustomFields] nvarchar(100)
'{CountryOfManufacture:$.PurchaseOrders.CountryOfManufacture , Range:
$.PurchaseOrders.Range}',
  SearchDetails nvarchar(max) 'USB food flash drive - chocolate bar ',
   LastEditedBy int '1'
   )


INSERT INTO [WideWorldImporters].[Purchasing].[PurchaseOrders]
SELECT *
FROM OPENJSON(@jsonda)
WITH(
```

```
PurchaseOrderID int '999',
SupplierID int '$.PurchaseOrders.Supplier' ,
OrderDate date '$.PurchaseOrders.OrderDate',
DeliveryMethodID int '1',
ContactPersonID int '101',
ExpectedDeliveryDate date '$.PurchaseOrders.ExpectedDeliveryDate',
SupplierReference nvarchar(20) '$.PurchaseOrders.SupplierReference',
 IsOrderFinalized bit  '0',
  LastEditedBy int '1',
LastEditedWhen datetime2(7) '2013-01-02 07:00:00.0000000'
);
```

```
INSERT INTO [WideWorldImporters].[Purchasing].[PurchaseOrderLines]
SELECT *
FROM OPENJSON(@jsonda)
WITH(
PurchaseOrderLineID int '999' ,
PurchaseOrderID int '999',
StockItemID int '999',
OrderedOuters int '999',
Description nvarchar(100) 'description',
ReceivedOuters int '999',
PackageTypeID int '999',
ExpectedUnitPricePerOuter decimal(18,2) '$.PurchaseOrders.UnitPrice',
IsOrderLineFinalized bit,
LastEditedBy int '1',
LastEditedWhen datetime2(7) '2013-01-02 07:00:00.0000000'
);
```

25. Revisit your answer in (19). Convert the result in JSON string and save it to the server using TSQL FOR JSON PATH.

```
DECLARE @col as nvarchar(max);
DECLARE @query as nvarchar(max);


SELECT @col = COALESCE(@col + ', ', '') + QUOTENAME([StockGroupName])
```

```
FROM (SELECT DISTINCT StockGroupName FROM Warehouse.StockGroups) a


set @query = 'select orderyear,' + @col + '
FROM (select StockGroupName,  Year(OrderDate)  as orderyear  , Quantity
FROM (select * from [WideWorldImporters].[Sales].[Orders]
where Year(OrderDate) in (2013, 2014, 2015 , 2016, 2017) ) o
join [WideWorldImporters].[Sales].[OrderLines] orl
on orl.OrderID = o.OrderID
join [WideWorldImporters].[Warehouse].[StockItemStockGroups] ssg
on orl.StockItemID = ssg.StockItemID
JOIN Warehouse.StockGroups sgs
ON ssg.StockGroupID = sgs.StockGroupID
) tempt
PIVOT(
sum(Quantity) for StockGroupName in (' + @col + ')
) pvt'

exec(@query)

DECLARE @tabb AS nvarchar(max)
select @tabb = 'create view vieww as '+ @query
EXEC(@tabb)

select *
from vieww
for JSON PATH, ROOT('Quantitysum')
```

26. Revisit your answer in (19). Convert the result into an XML string and save it to the server using TSQL FOR XML PATH.

```
DECLARE @col as nvarchar(max);
DECLARE @query as nvarchar(max);


SELECT @col = COALESCE(@col + ', ', '') + QUOTENAME([StockGroupName])
FROM (SELECT DISTINCT StockGroupName FROM Warehouse.StockGroups) a

set @query = 'select orderyear,' + @col + '
FROM (select StockGroupName,  Year(OrderDate)  as orderyear  , Quantity
FROM (select * from [WideWorldImporters].[Sales].[Orders]
```

```
where Year(OrderDate) in (2013, 2014, 2015 , 2016, 2017) ) o
join [WideWorldImporters].[Sales].[OrderLines] orl
on orl.OrderID = o.OrderID
join [WideWorldImporters].[Warehouse].[StockItemStockGroups] ssg
on orl.StockItemID = ssg.StockItemID
JOIN Warehouse.StockGroups sgs
ON ssg.StockGroupID = sgs.StockGroupID
) tempt
PIVOT(
sum(Quantity) for StockGroupName in (' + @col + ')
) pvt'

exec(@query)


DECLARE @tabb AS nvarchar(max)
select @tabb = 'create view vieww as '+ @query
EXEC(@tabb)


select orderyear ,
[Airline Novelties] as [AirlineNovelties],
[Clothing] AS [Clothing],
[Computing Novelties] as [ComputingNovelties],
[Furry Footwear] as [FurryFootwear],
[Mugs] as [mug],
[Novelty Items] as [NoveltyItems],
[Packaging Materials] AS [PackagingMaterials],
[Toys] as 'Toys',
[T-shirts] as [T-shirts],
[USB Novelties] AS [USBNovelties]
from vieww
FOR XML PATH
```

27. **Create a new table called ods.ConfirmedDeviveryJson with 3 columns (id, date, value) . Create a stored procedure, input is a date. The logic would load invoice information (all columns) as well as invoice line information (all columns) and forge them into a JSON string and then insert into the new table just created. Then write a query to run the stored procedure for each DATE that customer id 1 got something delivered to him.**

```
CREATE TABLE ods.ConfirmedDeviveryJson(
id INT NOT NULL PRIMARY KEY,
data date,
value nvarchar(max)
)

CREATE PROCEDURE procee
@DATEE DATE
AS
	DECLARE @js nvarchar(max);

	set @js = (SELECT i.*, il.InvoiceLineID, il.StockItemID, il.Description,
	il.PackageTypeID,
	il.Quantity, il.UnitPrice, il.TaxRate, il.TaxAmount,
	il.LineProfit,il.ExtendedPrice,
	il.LastEditedBy AS [InvoiceLineLastEditedBy], il.LastEditedWhen AS
	[InvoiceLineLastEditedWhen]
	FROM Sales.Invoices i
	JOIN Sales.InvoiceLines il
	ON i.InvoiceID = il.InvoiceID
	WHERE CONVERT(DATE, ConfirmedDeliveryTime )   =   @DATEE
		AND
		 i.CustomerID = 1
		FOR JSON PATH );

	INSERT INTO ods.ConfirmedDeviveryJson

	SELECT * FROM OPENJSON(@js)
	WITH(
	id INT '$.InvoiceID',
	data date '$.InvoiceDate',
	[value] nvarchar(max) AS JSON
	);

	GO


EXE procee @data  = ‘2013-01-02’;
```

28. Write a short essay talking about your understanding of transactions, locks and isolation levels.

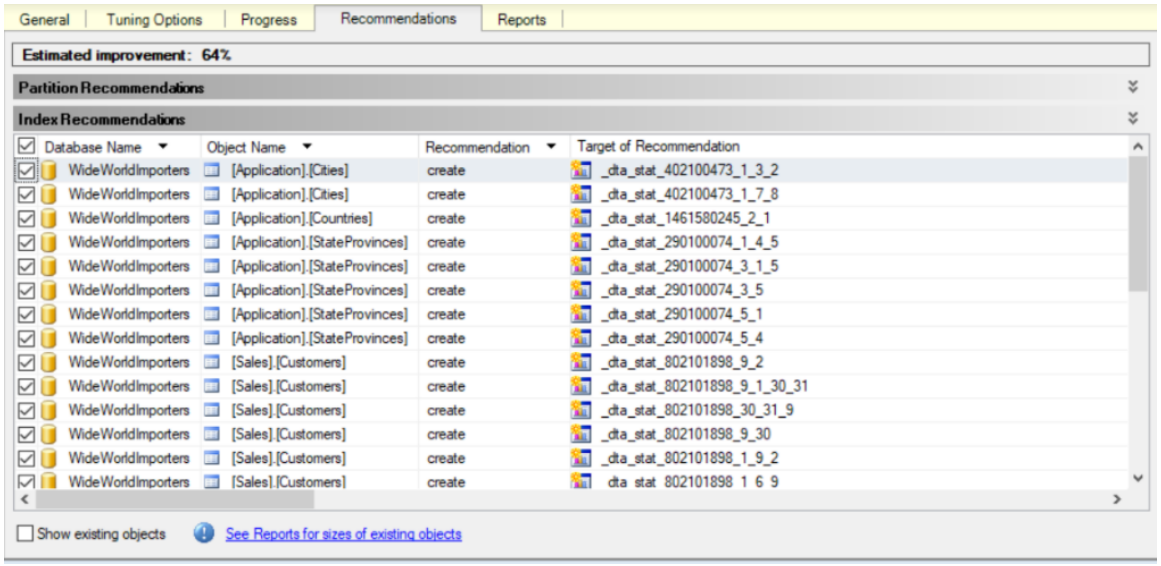
The design of isolation level is mainly to solve concurrency and security issues when executing transactions. Different isolation levels indicate the separation of data resources when read, modify and commit the changes in each transaction. There are four kinds of isolation level, read uncommitted, read committed,repeatable read and serializable. And locks are leveraged to fulfill the isolation among transactions. There are mainly three locks, shared lock, update lock and exclusive lock. Shared lock is granted when a transaction requests permission to read data . When shared lock is active, the locked data resources could only be read but could only be modified. After the lock being leased , data is available for other transactions. When an exclusive lock is active, the data could not be modified or read by other transactions till it is released. Update lock is mainly used to avoid deadlocks and only used in transactions to do with manipulation. Read uncommitted isolation level has the lowest level and is not restricted by the locks. Read committed isolation level which is the default level will issue shared lock but the lock is released after finishing reading data. Exclusive lock is maintained till the end of the transaction. For repeatable reads isolation level, once the shared lock and exclusive lock are issued , both of them will be hold till the end of the transaction. Serialiable is the highest isolation level which maintains the read and write lock til the end of transaction. Besides, it will also request the range lock when using the ranged where clause. which could prevent phantom reads. These lock based rules are mainly designed for concurrency control.

29. Write a short essay, plus screenshots talking about performance tuning in SQL Server. Must include Tuning Advisor, Extended Events, DMV, Logs and Execution Plan.

Tuning Advisor analyzes workloads to recommend indexes or partitioning strategies that will improve server's query performance. Here I used the Query Store as a workload. The extended events is a lightweight monitoring system and could help to collect different events and system activity for further analysis. DMV refers to the dynamic management view. It would return the information to

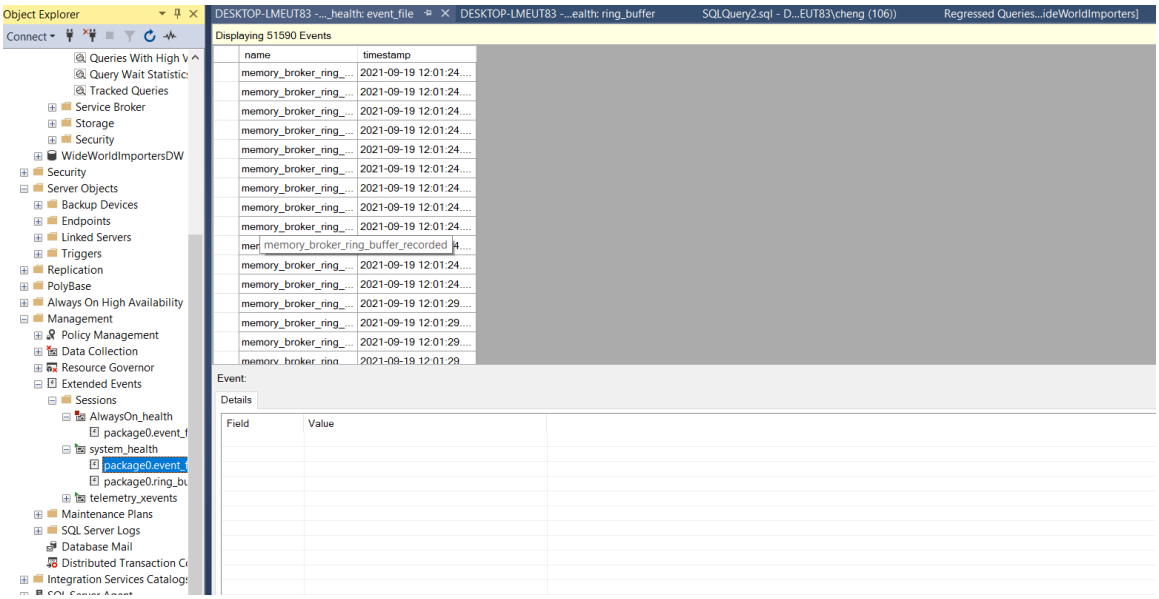
do with server state. Logs would record all the history transactions and related database modification. For each query in sql server, a execution plan would be generated to track the actual operations and execution steps taken in this transaction. The resources cost would also be recorded.

[Tuning Advisor]

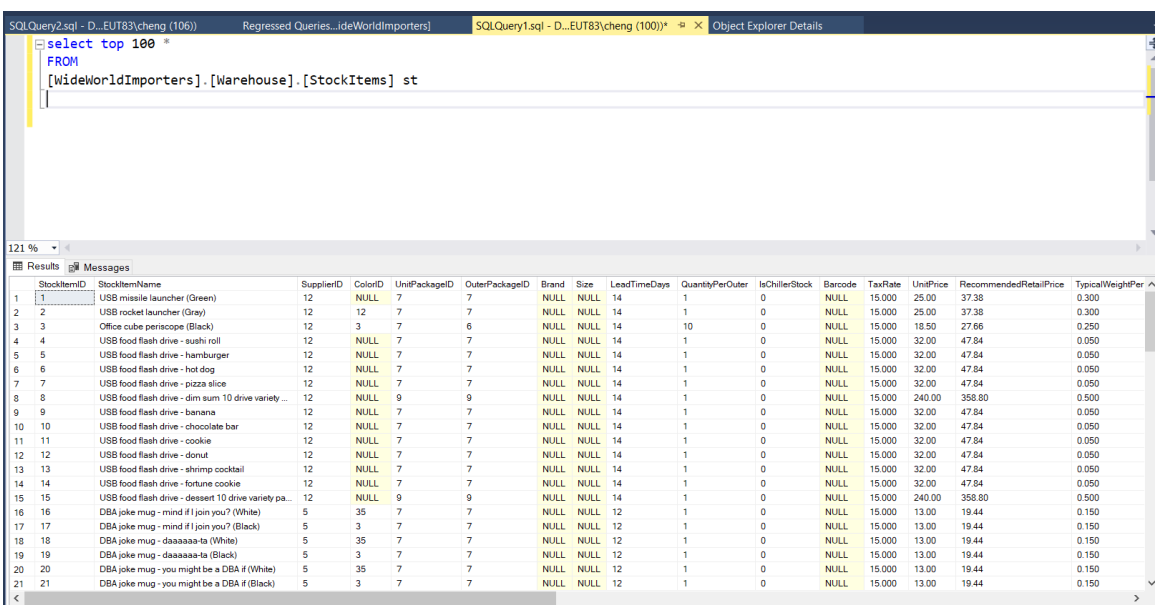


Database Name	Object Name	Recommendation	Target of Recommendation
WideWorldImporters	[Application].[Cities]	create	_dta_stat_402100473_1_3_2
WideWorldImporters	[Application].[Cities]	create	_dta_stat_402100473_1_7_8
WideWorldImporters	[Application].[Countries]	create	_dta_stat_1461580245_2_1
WideWorldImporters	[Application].[StateProvinces]	create	_dta_stat_290100074_1_4_5
WideWorldImporters	[Application].[StateProvinces]	create	_dta_stat_290100074_3_1_5
WideWorldImporters	[Application].[StateProvinces]	create	_dta_stat_290100074_3_5
WideWorldImporters	[Application].[StateProvinces]	create	_dta_stat_290100074_5_1
WideWorldImporters	[Application].[StateProvinces]	create	_dta_stat_290100074_5_4
WideWorldImporters	[Sales].[Customers]	create	_dta_stat_802101898_9_2
WideWorldImporters	[Sales].[Customers]	create	_dta_stat_802101898_9_1_30_31
WideWorldImporters	[Sales].[Customers]	create	_dta_stat_802101898_30_31_9
WideWorldImporters	[Sales].[Customers]	create	_dta_stat_802101898_9_30
WideWorldImporters	[Sales].[Customers]	create	_dta_stat_802101898_1_9_2
WideWorldImporters	[Sales].[Customers]	create	dta_stat_802101898_1_6_9

[Extended Event]

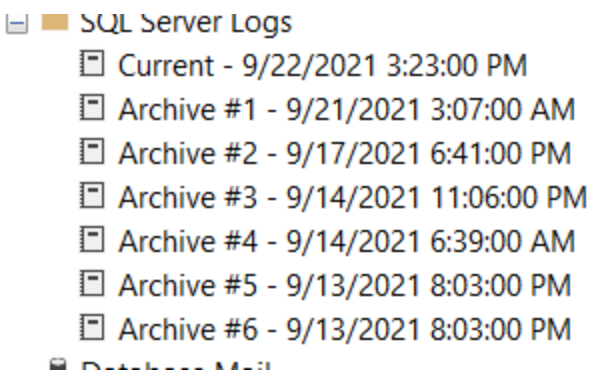


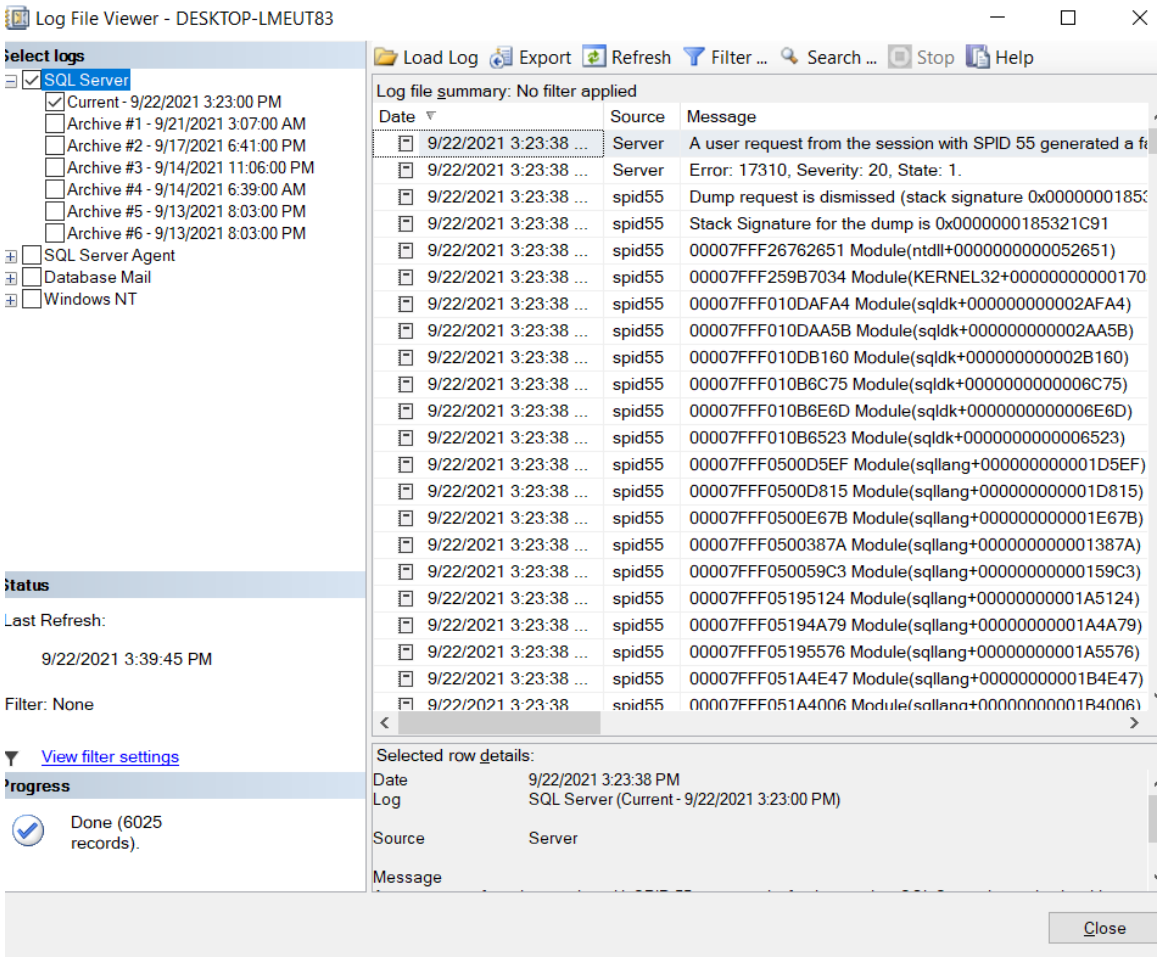
[DMV]



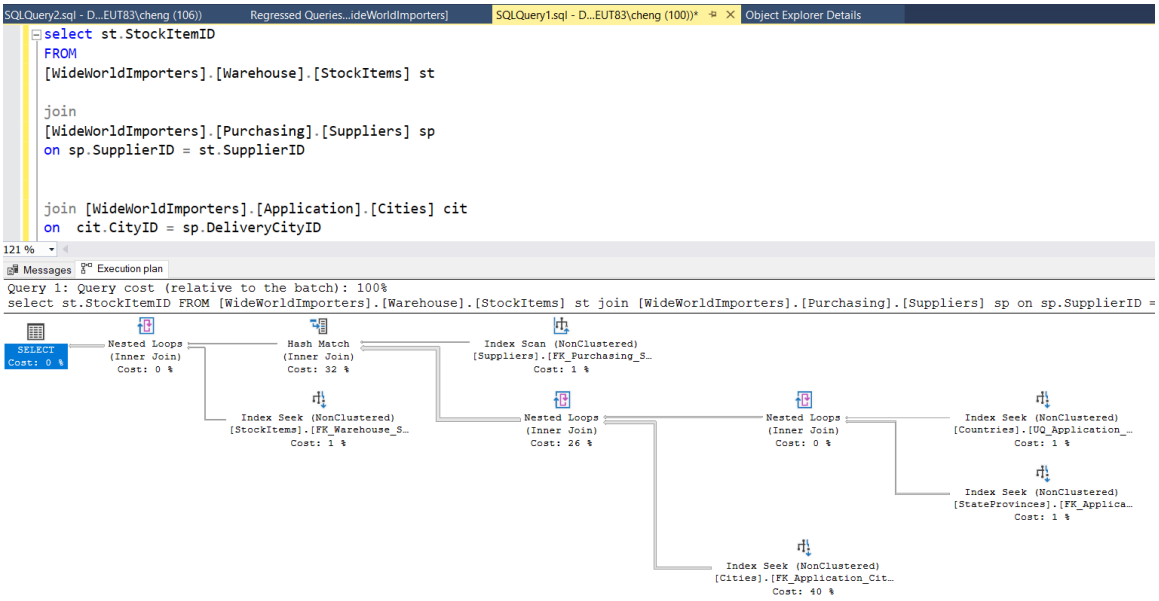
	StockItemID	StockItemName	SupplierID	ColorID	UnitPackageID	OuterPackageID	Brand	Size	LeadTimeDays	QuantityPerOuter	IsChillerStock	Barcode	TaxRate	UnitPrice	RecommendedRetailPrice	TypicalWeightPer
1	1	USB missile launcher (Green)	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	25.00	37.38	0.300
2	2	USB rocket launcher (Gray)	12	12	7	7	NULL	NULL	14	1	0	NULL	15.000	25.00	37.38	0.300
3	3	Office cube periscope (Black)	12	3	7	6	NULL	NULL	14	10	0	NULL	15.000	18.50	27.66	0.250
4	4	USB food flash drive - sushi roll	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
5	5	USB food flash drive - hamburger	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
6	6	USB food flash drive - hot dog	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
7	7	USB food flash drive - pizza slice	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
8	8	USB food flash drive - dim sum 10 drive variety ...	12	NULL	9	9	NULL	NULL	14	1	0	NULL	15.000	240.00	358.80	0.500
9	9	USB food flash drive - banana	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
10	10	USB food flash drive - chocolate bar	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
11	11	USB food flash drive - cookie	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
12	12	USB food flash drive - donut	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
13	13	USB food flash drive - shrimp cocktail	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
14	14	USB food flash drive - fortune cookie	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84	0.050
15	15	USB food flash drive - dessert 10 drive variety pa...	12	NULL	9	9	NULL	NULL	14	1	0	NULL	15.000	240.00	358.80	0.500
16	16	DBA joke mug - mind if I join you? (White)	5	35	7	7	NULL	NULL	12	1	0	NULL	15.000	13.00	19.44	0.150
17	17	DBA joke mug - mind if I join you? (Black)	5	3	7	7	NULL	NULL	12	1	0	NULL	15.000	13.00	19.44	0.150
18	18	DBA joke mug - daaaaaa-ta (White)	5	35	7	7	NULL	NULL	12	1	0	NULL	15.000	13.00	19.44	0.150
19	19	DBA joke mug - daaaaaa-ta (Black)	5	3	7	7	NULL	NULL	12	1	0	NULL	15.000	13.00	19.44	0.150
20	20	DBA joke mug - you might be a DBA if (White)	5	35	7	7	NULL	NULL	12	1	0	NULL	15.000	13.00	19.44	0.150
21	21	DBA joke mug - you might be a DBA if (Black)	5	3	7	7	NULL	NULL	12	1	0	NULL	15.000	13.00	19.44	0.150

[logs]


SQL Server Logs
Current - 9/22/2021 3:23:00 PM
Archive #1 - 9/21/2021 3:07:00 AM
Archive #2 - 9/17/2021 6:41:00 PM
Archive #3 - 9/14/2021 11:06:00 PM
Archive #4 - 9/14/2021 6:39:00 AM
Archive #5 - 9/13/2021 8:03:00 PM
Archive #6 - 9/13/2021 8:03:00 PM


Log File Viewer - DESKTOP-LMEUT83
Select logs
SQL Server
Current - 9/22/2021 3:23:00 PM
Archive #1 - 9/21/2021 3:07:00 AM
Archive #2 - 9/17/2021 6:41:00 PM
Archive #3 - 9/14/2021 11:06:00 PM
Archive #4 - 9/14/2021 6:39:00 AM
Archive #5 - 9/13/2021 8:03:00 PM
Archive #6 - 9/13/2021 8:03:00 PM
SQL Server Agent
Database Mail
Windows NT
Load Log
Export
Refresh
Filter ...
Search ...
Stop
Help
Log file summary: No filter applied
Date
Source
Message
9/22/2021 3:23:38 ... Server A user request from the session with SPID 55 generated a fa
9/22/2021 3:23:38 ... Server Error: 17310, Severity: 20, State: 1.
9/22/2021 3:23:38 ... spid55 Dump request is dismissed (stack signature 0x0000000185
9/22/2021 3:23:38 ... spid55 Stack Signature for the dump is 0x0000000185321C91
9/22/2021 3:23:38 ... spid55 00007FFF26762651 Module(ntdll+0000000000052651)
9/22/2021 3:23:38 ... spid55 00007FFF259B7034 Module(KERNEL32+00000000000170
9/22/2021 3:23:38 ... spid55 00007FFF010DAFA4 Module(sqldk+000000000002AFA4)
9/22/2021 3:23:38 ... spid55 00007FFF010DAA5B Module(sqldk+000000000002AA5B)
9/22/2021 3:23:38 ... spid55 00007FFF010DB160 Module(sqldk+000000000002B160)
9/22/2021 3:23:38 ... spid55 00007FFF010B6C75 Module(sqldk+0000000000006C75)
9/22/2021 3:23:38 ... spid55 00007FFF010B6E6D Module(sqldk+0000000000006E6D)
9/22/2021 3:23:38 ... spid55 00007FFF010B6523 Module(sqldk+0000000000006523)
9/22/2021 3:23:38 ... spid55 00007FFF0500D5EF Module(sqllang+000000000001D5EF)
9/22/2021 3:23:38 ... spid55 00007FFF0500D815 Module(sqllang+000000000001D815)
9/22/2021 3:23:38 ... spid55 00007FFF0500E67B Module(sqllang+000000000001E67B)
9/22/2021 3:23:38 ... spid55 00007FFF0500387A Module(sqllang+000000000001387A)
9/22/2021 3:23:38 ... spid55 00007FFF050059C3 Module(sqllang+00000000000159C3)
9/22/2021 3:23:38 ... spid55 00007FFF05195124 Module(sqllang+00000000001A5124)
9/22/2021 3:23:38 ... spid55 00007FFF05194A79 Module(sqllang+00000000001A4A79)
9/22/2021 3:23:38 ... spid55 00007FFF05195576 Module(sqllang+00000000001A5576)
9/22/2021 3:23:38 ... spid55 00007FFF051A4E47 Module(sqllang+00000000001B4E47)
Status
Last Refresh:
9/22/2021 3:39:45 PM
Filter: None
View filter settings
Progress
Done (6025 records).
Selected row details:
Date 9/22/2021 3:23:38 PM
Log SQL Server (Current - 9/22/2021 3:23:00 PM)
Source Server
Message
Close

[Execution Plan]



Assignments 30 - 32 are group assignments.

30. **Write a short essay talking about a scenario: Good news everyone! We (Wide World Importers) just brought out a small company called "Adventure works"! Now that bike shop is our sub-company. The first thing of all works pending would be to merge the user logon information, person information (including emails, phone numbers) and products (of course, add category, colors) to WWI database. Include screenshot, mapping and query.**

Moving person and user logon information:

INSERT INTO WideWorldImporters.Application.People(FullName, PreferredName, IsPermittedToLogon, LogonName, IsExternalLogonProvider, HashedPassword, IsSystemUser, IsEmployee, IsSalesperson, UserPreferences, PhoneNumber, FaxNumber, EmailAddress, Photo, CustomFields, LastEditedBy)
SELECT CONCAT(p.FirstName, ' ', p.MiddleName, ' ', p.LastName) [FullName], FirstName [PreferredName],
CASE WHEN e.LoginID IS NOT NULL THEN 1 ELSE 0 END [IsPermittedToLogon], ISNULL(e.LoginID, 'NO LOGON') [LogonName], 0 [IsExternalLogonProvider],
CONVERT(VARBINARY(256), pw.PasswordHash) [HashedPassword], 0 [IsSystemUser], CASE WHEN p.PersonType IN ('SC', 'SP', 'EM') THEN 1 ELSE 0 END [IsEmployee],
CASE WHEN p.PersonType = 'SP' THEN 1 ELSE 0 END [IsSalesperson], NULL [UserPreferences], pp.PhoneNumber [PhoneNumber], NULL [FaxNumber],
ea.EmailAddress [EmailAddress], NULL [Photo], CONCAT('{ "OtherLanguages": [] ,"HireDate":"', e.HireDate, '","Title":"', e.JobTitle, '"}') [CustomFields],
1 [LastEditedBy]
FROM AdventureWorks2019.Person.Person p
LEFT JOIN AdventureWorks2019.HumanResources.Employee e
ON p.BusinessEntityID = e.BusinessEntityID
JOIN AdventureWorks2019.Person.Password pw
ON p.BusinessEntityID = pw.BusinessEntityID
JOIN AdventureWorks2019.Person.EmailAddress ea
ON p.BusinessEntityID = ea.BusinessEntityID
JOIN AdventureWorks2019.Person.PersonPhone pp
ON p.BusinessEntityID = pp.BusinessEntityID

```
INSERT INTO WideWorldImporters.Application.People(FullName, PreferredName, IsPermittedToLogon, LogonName, IsExternalLogonProvider, HashedPassword,
IsSystemUser, IsEmployee, IsSalesperson, UserPreferences, PhoneNumber, FaxNumber, EmailAddress, Photo, CustomFields, LastEditedBy)
SELECT CONCAT(p.FirstName, ' ', p.MiddleName, ' ', p.LastName) [FullName], FirstName [PreferredName],
  CASE WHEN e.LoginID IS NOT NULL THEN 1 ELSE 0 END [IsPermittedToLogon], ISNULL(e.LoginID, 'NO LOGON') [LogonName], 0 [IsExternalLogonProvider],
  CONVERT(VARBINARY(256), pw.PasswordHash) [HashedPassword], 0 [IsSystemUser], CASE WHEN p.PersonType IN ('SC', 'SP', 'EM') THEN 1 ELSE 0 END [IsEmployee],
  CASE WHEN p.PersonType = 'SP' THEN 1 ELSE 0 END [IsSalesperson], NULL [UserPreferences], pp.PhoneNumber [PhoneNumber], NULL [FaxNumber],
  ea.EmailAddress [EmailAddress], NULL [Photo], CONCAT('{ "OtherLanguages": [] ,"HireDate":"', e.HireDate, '","Title":"', e.JobTitle, '"}') [CustomFields],
  1 [LastEditedBy]
FROM AdventureWorks2019.Person.Person p
LEFT JOIN AdventureWorks2019.HumanResources.Employee e
ON p.BusinessEntityID = e.BusinessEntityID
JOIN AdventureWorks2019.Person.Password pw
ON p.BusinessEntityID = pw.BusinessEntityID
JOIN AdventureWorks2019.Person.EmailAddress ea
ON p.BusinessEntityID = ea.BusinessEntityID
JOIN AdventureWorks2019.Person.PersonPhone pp
ON p.BusinessEntityID = pp.BusinessEntityID
```

Messages

```
(19972 rows affected)

Completion time: 2021-09-22T15:53:34.5808287-04:00
```

```
SELECT * FROM Application.People
```

100 %

Results | Messages

	PersonID	FullName	PreferredName	SearchName	IsPermittedToLogon	LogonName	IsExternalLogonProvider	HashedPassword
1	1	Data Conversion Only	Data Conversion Only	Data Conversion Only Data Conversion Only	0	NO LOGON	0	NULL
2	2	Kayla Woodcock	Kayla	Kayla Kayla Woodcock	1	kaylaw@wideworldimporters.com	0	0x616E9B558976525E7F14D780EBAE80C68586958DC9
3	3	Hudson Onslow	Hudson	Hudson Hudson Onslow	1	hudsono@wideworldimporters.com	0	0x23668CCC579015EA934736C3D7B87E86360EB5EEE1
4	4	Isabella Rupp	Isabella	Isabella Isabella Rupp	1	isabellar@wideworldimporters.com	0	0xB45E7C4E37C32FA9A5A3161B9DB1C9C1E787BB7DB4
5	5	Eva Muirden	Eva	Eva Eva Muirden	0	evam@wideworldimporters.com	0	0xE682D36E43B6A3940ED6428B2DE3CEEDD1763C5E0
6	6	Sophia Hinton	Sophia	Sophia Sophia Hinton	1	sophiah@wideworldimporters.com	0	0x451BB10A515F06331540DB392031F9D9BC4EF536A1F
7	7	Amy Trefl	Amy	Amy Amy Trefl	1	amyt@wideworldimporters.com	0	0x7A92BBEA830C5ED027DCC1D710130EED9EA50FB3E
8	8	Anthony Grosse	Anthony	Anthony Anthony Grosse	1	anthonyg@wideworldimporters.com	0	0x2FD8B838A3C77778C990F464073AA23C0EEE01976 3E
9	9	Alica Fatnowna	Alica	Alica Alica Fatnowna	1	alicaf@wideworldimporters.com	0	0x7DFAB08E9AC574C5B15CF19D18E5B3EB466EAC7392
10	10	Stella Rosenhain	Stella	Stella Stella Rosenhain	1	stellar@wideworldimporters.com	0	0x1BA4B55887E2BDCB06087A20E1CC608ADDCA538BAE
11	11	Ethan Onslow	Ethan	Ethan Ethan Onslow	1	ethano@wideworldimporters.com	0	0xD70F37F5C019499959DDF987E5343B957FEB58959A
12	12	Henry Forlonge	Henry	Henry Henry Forlonge	1	henryf@wideworldimporters.com	0	0x3F74BAD95BD9059EFCF80F983899E24369959FD488
13	13	Hudson Hollinworth	Hudson	Hudson Hudson Hollinworth	1	hudsonh@wideworldimporters.com	0	0x4AC0A24180C54F425AC88CA33B62136A37B6AAA1943
14	14	Lily Code	Lily	Lily Lily Code	1	lilyc@wideworldimporters.com	0	0xD00658893B3F96277088B3C71DCFBF4DF6E3947EEC
15	15	Taj Shand	Taj	Taj Taj Shand	1	tajs@wideworldimporters.com	0	0x9AEFEEC0DBB0EA6B25F0EB99C62C724F18428D457
16	16	Archer Lamble	Archer	Archer Archer Lamble	0	archerl@wideworldimporters.com	0	0x06187F68631295411B022C48B07338F20F4C5C73B31

Moving product group information:

INSERT INTO WideWorldImporters.Warehouse.StockGroups(StockGroupName, LastEditedBy)
SELECT pc.Name [StockGroupName], 1 [LastEditedBy]
FROM AdventureWorks2019.Production.ProductCategory pc
WHERE NOT EXISTS
*(SELECT * FROM WideWorldImporters.Warehouse.StockGroups*
WHERE StockGroupName = pc.Name COLLATE SQL_Latin1_General_CP1_CI_AS);

```
INSERT INTO WideWorldImporters.Warehouse.StockGroups(StockGroupName, LastEditedBy)
SELECT pc.Name [StockGroupName], 1 [LastEditedBy]
FROM AdventureWorks2019.Production.ProductCategory pc
WHERE NOT EXISTS
(SELECT * FROM WideWorldImporters.Warehouse.StockGroups
WHERE StockGroupName = pc.Name COLLATE SQL_Latin1_General_CP1_CI_AS);
```

0 %

Messages

```
(3 rows affected)

Completion time: 2021-09-22T16:09:49.3727675-04:00
```

```
SELECT * FROM Warehouse.StockGroups
```

100 %

Results | Messages

	StockGroupID	StockGroupName	LastEditedBy	ValidFrom	ValidTo
1	1	Novelty Items	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
2	2	Clothing	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
3	3	Mugs	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
4	4	T-Shirts	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
5	5	Airline Novelties	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
6	6	Computing Novelties	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
7	7	USB Novelties	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
8	8	Furry Footwear	9	2016-01-01 16:00:00.0000000	9999-12-31 23:59:59.9999999
9	9	Toys	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
10	10	Packaging Materials	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
11	11	Accessories	1	2021-09-22 20:09:49.3588240	9999-12-31 23:59:59.9999999
12	12	Bikes	1	2021-09-22 20:09:49.3588240	9999-12-31 23:59:59.9999999
13	13	Components	1	2021-09-22 20:09:49.3588240	9999-12-31 23:59:59.9999999

Moving color information:

INSERT INTO WideWorldImporters.Warehouse.Colors(ColorName, LastEditedBy)
SELECT DISTINCT Color, 1
FROM AdventureWorks2019.Production.Product p
WHERE p.Color IS NOT NULL AND NOT EXISTS
(SELECT * FROM WideWorldImporters.Warehouse.Colors c
WHERE c.ColorName = p.Color COLLATE SQL_Latin1_General_CP1_CI_AS)

```
INSERT INTO WideWorldImporters.Warehouse.Colors(ColorName, LastEditedBy)
SELECT DISTINCT Color, 1
FROM AdventureWorks2019.Production.Product p
WHERE p.Color IS NOT NULL AND NOT EXISTS
(SELECT * FROM WideWorldImporters.Warehouse.Colors c
WHERE c.ColorName = p.Color COLLATE SQL_Latin1_General_CP1_CI_AS)
```

00 %

Messages

```
(3 rows affected)

Completion time: 2021-09-22T16:16:54.7084886-04:00
```

```
SELECT * FROM Warehouse.Colors
```

100 %

Results | Messages

	ColorID	ColorName	LastEditedBy	ValidFrom	ValidTo
23	23	Olive	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
24	24	Orange	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
25	25	Plum	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
26	26	Puce	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
27	27	Purple	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
28	28	Red	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
29	29	Royal Blue	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
30	30	Salmon	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
31	31	Silver	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
32	32	Tan	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
33	33	Teal	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
34	34	Wheat	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999
35	35	White	1	2013-01-01 00:00:00.0000000	9999-12-31 23:59:59.9999999

Moving vendor information:

INSERT INTO WideWorldImporters.Purchasing.Suppliers(SupplierName, SupplierCategoryID, PrimaryContactPersonID, AlternateContactPersonID, DeliveryMethodID, DeliveryCityID, PostalCityID, PaymentDays, BankAccountNumber, PhoneNumber, FaxNumber, WebsiteURL, DeliveryAddressLine1,
DeliveryPostalCode, PostalAddressLine1, PostalPostalCode, LastEditedBy)
SELECT v.Name, 1 [SupplierCategoryID], 1 [PrimaryContactPersonID], 1 [AlternateContactPersonID], 1 [DeliveryMethodID], 1 [DeliveryCityID], 1 [PostalCityID], 0 [PaymentDays], v.AccountNumber [BankAccountNumber], '' [PhoneNumber], '' [FaxNumber], '' [WebsiteURL], '' [DeliveryAddressLine1], '' [DeliveryPostalCode], '' [PostalAddressLine1], '' [PostalPostalCode], 1 [LastEditedBy]

FROM AdventureWorks2019.Purchasing.Vendor v
*WHERE NOT EXISTS (SELECT * FROM WideWorldImporters.Purchasing.Suppliers s*
WHERE s.SupplierName = v.Name COLLATE SQL_Latin1_General_CP1_CI_AS)

```
INSERT INTO WideWorldImporters.Purchasing.Suppliers(SupplierName, SupplierCategoryID, PrimaryContactPersonID, AlternateContactPersonID,
DeliveryMethodID, DeliveryCityID, PostalCityID, PaymentDays, BankAccountNumber, PhoneNumber, FaxNumber, WebsiteURL, DeliveryAddressLine1,
DeliveryPostalCode, PostalAddressLine1, PostalPostalCode, LastEditedBy)
SELECT v.Name, 1 [SupplierCategoryID], 1 [PrimaryContactPersonID], 1 [AlternateContactPersonID], 1 [DeliveryMethodID], 1 [DeliveryCityID],
1 [PostalCityID], 0 [PaymentDays], v.AccountNumber [BankAccountNumber], '' [PhoneNumber], '' [FaxNumber], '' [WebsiteURL], '' [DeliveryAddressLine1],
'' [DeliveryPostalCode], '' [PostalAddressLine1], '' [PostalPostalCode], 1 [LastEditedBy]
FROM AdventureWorks2019.Purchasing.Vendor v
WHERE NOT EXISTS (SELECT * FROM WideWorldImporters.Purchasing.Suppliers s WHERE s.SupplierName = v.Name COLLATE SQL_Latin1_General_CP1_CI_AS)
```

100 %

Messages

```
(102 rows affected)

Completion time: 2021-09-22T16:35:37.4707584-04:00
```

```
SELECT * FROM Purchasing.Suppliers
```

100 %

Results | Spatial results | Messages

	SupplierID	SupplierName	SupplierCategoryID	PrimaryContactPersonID	AlternateContactPersonID	DeliveryMethodID	DeliveryCityID	PostalCityID	SupplierReference	BankAccountName	BankAccountBranch
1	1	A Datum Corporation	2	21	22	7	38171	38171	AA20384	A Datum Corporation	Woodgrove Bank Zionsville
2	2	Contoso, Ltd.	2	23	24	9	13870	13870	B2084020	Contoso Ltd	Woodgrove Bank Greenbank
3	3	Consolidated Messenger	6	25	26	NULL	30378	30378	209340283	Consolidated Messenger	Woodgrove Bank San Francisco
4	4	Fabrikam, Inc.	4	27	28	7	18557	18557	293092	Fabrikam Inc	Woodgrove Bank Lakeview Heights
5	5	Graphic Design Institute	2	29	30	10	18634	18634	08803922	Graphic Design Institute	Woodgrove Bank Lanagan
6	6	Humongous Insurance	9	31	32	NULL	18656	18656	082420938	Humongous Insurance	Woodgrove Bank Lancing
7	7	Litware, Inc.	5	33	34	2	22602	22602	BC0280982	Litware Inc	Woodgrove Bank Mokelumne Hill
8	8	Lucerne Publishing	2	35	36	10	17161	17161	JQ082304802	Lucerne Publishing	Woodgrove Bank Jonesborough
9	9	Nod Publishers	2	37	38	10	10346	10346	GL08029802	Nod Publishers	Woodgrove Bank Elizabeth City
10	10	Northwind Electric Cars	3	39	40	8	7899	7899	ML0300202	Northwind Electric Cars	Woodgrove Bank Crandon Lakes
11	11	Trey Research	8	41	42	NULL	17277	17277	082304822	Trey Research	Woodgrove Bank Kadoka
12	12	The Phone Company	2	43	44	7	17346	17346	237408032	The Phone Company	Woodgrove Bank Karlstad
13	13	Woodgrove Bank	7	45	46	NULL	30378	30378	028034202	Woodgrove Bank	Woodgrove Bank San Francisco
14	15	Australia Bike Retailer	1	1	1	1	1	1	NULL	NULL	NULL
15	16	Allenson Cycles	1	1	1	1	1	1	NULL	NULL	NULL
16	17	Advanced Bicycles	1	1	1	1	1	1	NULL	NULL	NULL

Moving product information:

SELECT p.Name, s.SupplierID [SupplierID], c.ColorID [ColorID], 7 [UnitPackageID],
7 [OuterPackageID], NULL [Brand], p.Size [Size],
pv.AverageLeadTime [LeadTimeDays], 1 [QuantityPerOuter], 0 [IsChillerStock],
NULL [Barcode], 6.0 [TaxRate], p.ListPrice [UnitPrice],
pv.StandardPrice [RecommendedRetailPrice], ISNULL(p.Weight,0)
[TypicalWeightPerUnit], pd.Description [MarketingComments], pd.Description
[InternalComments],
pp.LargePhoto [Photo], NULL [CustomFields], 1 [LastEditedBy], ROW_NUMBER()
OVER(PARTITION BY p.ProductID ORDER BY p.Name) [Row]
INTO #productTemp
FROM AdventureWorks2019.Production.Product p
JOIN AdventureWorks2019.Purchasing.ProductVendor pv
ON p.ProductID = pv.ProductID
JOIN AdventureWorks2019.Purchasing.Vendor v
ON pv.BusinessEntityID = v.BusinessEntityID
JOIN WideWorldImporters.Purchasing.Suppliers s
ON v.Name = s.SupplierName COLLATE SQL_Latin1_General_CP1_CI_AS
JOIN AdventureWorks2019.Production.ProductModel pm
ON p.ProductModelID = pm.ProductModelID

JOIN AdventureWorks2019.Production.ProductModelProductDescriptionCulture pmpdc
ON pm.ProductModelID = pmpdc.ProductModelID
JOIN AdventureWorks2019.Production.ProductDescription pd
ON pmpdc.ProductDescriptionID = pd.ProductDescriptionID
JOIN AdventureWorks2019.Production.ProductProductPhoto ppp
ON p.ProductID = ppp.ProductID
JOIN AdventureWorks2019.Production.ProductPhoto pp
ON ppp.ProductPhotoID = pp.ProductPhotoID
JOIN WideWorldImporters.Warehouse.Colors c
ON p.Color = c.ColorName COLLATE SQL_Latin1_General_CP1_CI_AS
WHERE NOT EXISTS
*(SELECT * FROM WideWorldImporters.Warehouse.StockItems si*
WHERE si.StockItemName = p.Name COLLATE SQL_Latin1_General_CP1_CI_AS)

```
SELECT * FROM #productTemp
```

100 %

Results | Messages

	Name	SupplierID	ColorID	UnitPackageID	OuterPackageID	Brand	Size	LeadTimeDays	QuantityPerOuter	IsChillerStock	Barcode	TaxRate	UnitPrice	RecommendedRetailPrice	TypicalWeightPerUnit
1	Women's Mountain Shorts, S	105	3	7	7	NULL	S	25	1	0	NULL	6.0	69.99	25.45	0.00
2	Women's Mountain Shorts, S	105	3	7	7	NULL	S	25	1	0	NULL	6.0	69.99	25.45	0.00
3	Women's Mountain Shorts, S	105	3	7	7	NULL	S	25	1	0	NULL	6.0	69.99	25.45	0.00
4	Women's Mountain Shorts, M	105	3	7	7	NULL	M	25	1	0	NULL	6.0	69.99	25.45	0.00
5	Women's Mountain Shorts, M	105	3	7	7	NULL	M	25	1	0	NULL	6.0	69.99	25.45	0.00
6	Women's Mountain Shorts, M	105	3	7	7	NULL	M	25	1	0	NULL	6.0	69.99	25.45	0.00
7	Women's Mountain Shorts, M	105	3	7	7	NULL	M	25	1	0	NULL	6.0	69.99	25.45	0.00
8	Women's Mountain Shorts, M	105	3	7	7	NULL	M	25	1	0	NULL	6.0	69.99	25.45	0.00
9	Sport-100 Helmet, Red	29	28	7	7	NULL	NULL	30	1	0	NULL	6.0	34.99	13.25	0.00
10	Sport-100 Helmet, Red	29	28	7	7	NULL	NULL	30	1	0	NULL	6.0	34.99	13.25	0.00
11	Sport-100 Helmet, Red	29	28	7	7	NULL	NULL	30	1	0	NULL	6.0	34.99	13.25	0.00
12	Sport-100 Helmet, Red	29	28	7	7	NULL	NULL	30	1	0	NULL	6.0	34.99	13.25	0.00

INSERT INTO WideWorldImporters.Warehouse.StockItems(StockItemName, SupplierID, ColorID, UnitPackageID, OuterPackageID, Brand, Size, LeadTimeDays, QuantityPerOuter, IsChillerStock, Barcode, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, MarketingComments, InternalComments,
Photo, CustomFields, LastEditedBy)
SELECT Name+CAST(Row AS nvarchar(10)) [StockItemName], SupplierID, ColorID, UnitPackageID, OuterPackageID, Brand, Size, LeadTimeDays, QuantityPerOuter, IsChillerStock, Barcode, TaxRate, UnitPrice, RecommendedRetailPrice, TypicalWeightPerUnit, MarketingComments, InternalComments,
Photo, CustomFields, LastEditedBy
FROM #productTemp

```
SELECT * FROM Warehouse.StockItems
```

100 %

Results | Messages

	StockItemID	StockItemName	SupplierID	ColorID	UnitPackageID	OuterPackageID	Brand	Size	LeadTimeDays	QuantityPerOuter	IsChillerStock	Barcode	TaxRate	UnitPrice	RecommendedReta
1	1	USB missile launcher (Green)	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	25.00	37.38
2	2	USB rocket launcher (Gray)	12	12	7	7	NULL	NULL	14	1	0	NULL	15.000	25.00	37.38
3	3	Office cube periscope (Black)	12	3	7	6	NULL	NULL	14	10	0	NULL	15.000	18.50	27.66
4	4	USB food flash drive - sushi roll	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84
5	5	USB food flash drive - hamburger	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84
6	6	USB food flash drive - hot dog	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84
7	7	USB food flash drive - pizza slice	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84
8	8	USB food flash drive - dim sum 10 drive variety ...	12	NULL	9	9	NULL	NULL	14	1	0	NULL	15.000	240.00	358.80
9	9	USB food flash drive - banana	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84
10	10	USB food flash drive - chocolate bar	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84
11	11	USB food flash drive - cookie	12	NULL	7	7	NULL	NULL	14	1	0	NULL	15.000	32.00	47.84

```
INSERT INTO
WideWorldImporters.Warehouse.StockItemStockGroups(StockItemID,
StockGroupID, LastEditedBy)
SELECT si.StockItemID, ps.ProductCategoryID [StockGroupID], 1 [LastEditedBy]
FROM AdventureWorks2019.Production.Product p JOIN
AdventureWorks2019.Production.ProductSubcategory ps ON
p.ProductSubcategoryID = ps.ProductSubcategoryID
JOIN #productTemp pt
ON p.Name = pt.Name
JOIN WideWorldImporters.Warehouse.StockItems si
ON pt.Name+CAST(pt.Row AS nvarchar(10)) = si.StockItemName COLLATE
SQL_Latin1_General_CP1_CI_AS
```

```
SELECT * FROM Warehouse.StockItemStockGroups
```

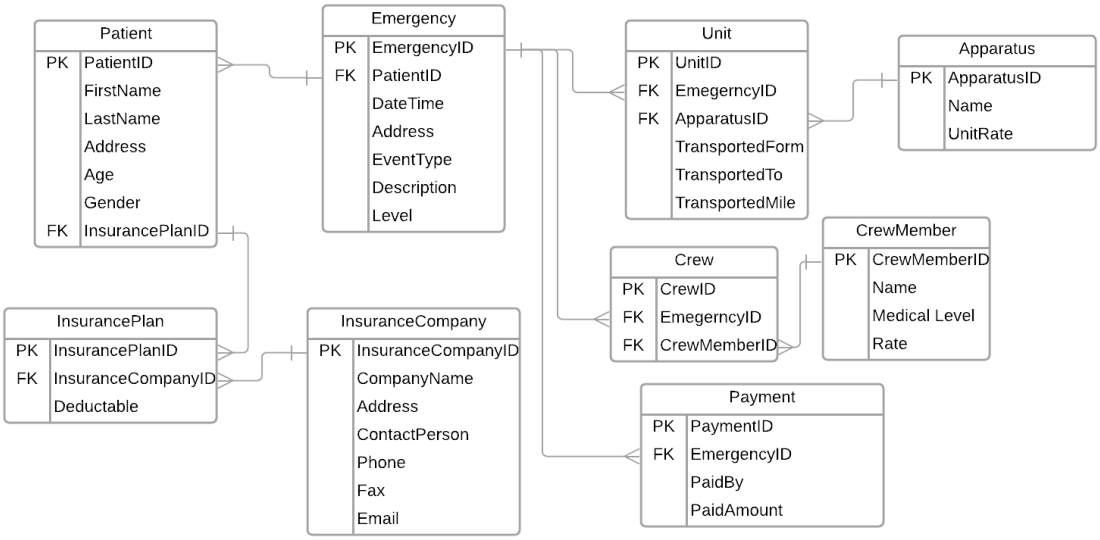
100 %

Results | Messages

	StockItemStockGroupID	StockItemID	StockGroupID	LastEditedBy	LastEditedWhen
1	1	1	6	1	2013-01-01 00:00:00.0000000
2	2	1	1	1	2013-01-01 00:00:00.0000000
3	3	1	7	1	2013-01-01 00:00:00.0000000
4	4	2	6	1	2013-01-01 00:00:00.0000000
5	5	2	1	1	2013-01-01 00:00:00.0000000
6	6	2	7	1	2013-01-01 00:00:00.0000000
7	7	3	6	1	2013-01-01 00:00:00.0000000
8	8	3	1	1	2013-01-01 00:00:00.0000000
9	9	4	6	1	2013-01-01 00:00:00.0000000
10	10	4	1	1	2013-01-01 00:00:00.0000000
11	11	4	7	1	2013-01-01 00:00:00.0000000
12	12	5	6	1	2013-01-01 00:00:00.0000000
13	13	5	1	1	2013-01-01 00:00:00.0000000
14	14	5	7	1	2013-01-01 00:00:00.0000000
15	15	6	6	1	2013-01-01 00:00:00.0000000

31. **Database Design: OLTP db design request for EMS business: when people call 911 for medical emergency, 911 will dispatch UNITs to the given address. A UNIT means a crew on an apparatus (Fire Engine, Ambulance, Medic Ambulance, Helicopter, EMS supervisor). A crew member would have a medical level (EMR, EMT, A-EMT, Medic). All the treatments provided on scene are free.**

If the patient needs to be transported, that's where the bill comes in. A bill consists of Units dispatched (Fire Engine and EMS Supervisor are free), crew members provided care (EMRs and EMTs are free), Transported miles from the scene to the hospital (Helicopters have a much higher rate, as you can image) and tax (Tax rate is 6%). Bill should be sent to the patient insurance company first. If there is a deductible, we send the unpaid bill to the patient only. Don't forget about patient information, medical nature and bill paying status.



32. **Remember the discussion about those two databases from the class, also remember, those data models are not perfect. You can always add new columns (but not alter or drop columns) to any tables. Suggesting adding Ingested DateTime and Surrogate Key columns. Study the Wide World Importers DW. Think the integration schema is the ODS. Come up with a TSQL Stored Procedure driven solution to move the data from WWI database to ODS, and then from the ODS to the fact tables and dimension tables. By the way, WWI DW is a galaxy schema db. Requirements:**
    a. **Luckly, we only start with 1 fact: Order. Other facts can be ignored for now.**
    b. **Add a new dimension: Country of Manufacture. It should be given on top of Stock Items.**
    c. **Write script(s) and stored procedure(s) for the entire ETL from WWI db to DW.**

```
CREATE Procedure wwietl

AS

insert into [WideWorldImportersDW].[Integration].[Order_Staging]

select   NEWID(),ingestion_time(), dwcit.[City Key] , dwcu.[customer key],
dwsi.[stock item key], o.OrderDate, o.ExpectedDeliveryDate, dwemp.[Employee
Key] , PickedByPersonID,dwcu.[Customer Key] , o.OrderID, o.BackorderOrderID,
orl.Description, packt.PackageTypeName,

orl.Quantity,    orl.UnitPrice, orl.TaxRate, (orl.Quantity *
orl.UnitPrice)*(1-orl.TaxRate/100) as totalexcludingtax, (orl.Quantity *
orl.UnitPrice) * (orl.TaxRate/100) as taxamount,
(orl.Quantity * orl.UnitPrice)  as totalincludingtax, dwpurchase.[Lineage Key],
dwcit.[City Key],cu.CustomerID, orl.StockItemID, o.LastEditedWhen

FROM [WideWorldImporters].[Sales].[OrderLines] orl
JOIN     [WideWorldImporters].[Sales].[Orders] o
ON orl.OrderID = o.OrderID
JOIN [WideWorldImporters].[Sales].[Customers] c
on o.CustomerID = c.CustomerID
JOIN [WideWorldImporters].[Application].[Cities] cit
ON c.DeliveryCityID = cit.CityID
JOIN [WideWorldImporters].[Application].[StateProvinces] statee
ON statee.StateProvinceID = cit.StateProvinceID
JOIN [WideWorldImporters].[Application].[Countries] count
ON count.CountryID = statee.CountryID
JOIN [WideWorldImportersDW].[Dimension].[Customer] dwcu
ON dwcu.[WWI Customer ID] = c.CustomerID
JOIN [WideWorldImporters].[Sales].[Invoices] inv
ON inv.CustomerID = c.CustomerID
JOIN [WideWorldImportersDW].[Fact].[Sale] dwfs
on dwfs.[WWI Invoice ID]  = inv.InvoiceID
JOIN  [WideWorldImportersDW].[Dimension].[City] dwcit
ON dwcit.[WWI City ID] = cit.CityID
JOIN [WideWorldImportersDW].[Dimension].[Stock Item] dwsi
ON dwsi.[WWI Stock Item ID] = orl.StockItemID
JOIN  [WideWorldImportersDW].[Fact].[Purchase] dwpurchase
ON dwpurchase.[WWI Purchase Order ID]  =            o.OrderID
```

```
JOIN [WideWorldImportersDW].[Dimension].[Employee] dwemp
ON dwemp.[WWI Employee ID] =  o.SalespersonPersonID
JOIN [WideWorldImportersDW].[Dimension].[Customer] dwcu2
on o.PickedByPersonID = dwcu2.[Customer Key]
JOIN WideWorldImporters].[Warehouse].[PackageTypes] packt
on packt.PackageTypeID = orl.PackageTypeID;


INSERT INTO [WideWorldImportersDW].[Fact].[Order]
select [City Key],[Customer Key],[Stock Item Key],
                [Order Date Key],[Picked Date Key],[Salesperson Key],
                [Picker Key],[WWI Order ID],[WWI Backorder ID],[Description],
                [Package],[Quantity],[Unit Price],[Tax Rate],[Total Excluding Tax],
                [Tax Amount],[Total Including Tax],[Lineage Key]

FROM [WideWorldImportersDW].[Integration].[Order_Staging] ;



CREATE TABLE [WideWorldImportersDW].[Dimension].[CountryOfManufacture](
StockItemID int not null PRIMARY KEY,
StockItemName nvarchar(100),
CountryOfManufacture nvarchar(max) NULL
);

WITH table1 AS(
select si.StockItemID , si.StockItemName
,JSON_VALUE(si.CustomFields,'$.CountryOfManufacture') as
CountryOfManufacture
FROM [WideWorldImporters].[Warehouse].[StockItems] si

)

INSERT INTO [WideWorldImportersDW].[Dimension].[CountryOfManufacture]
SELECT * FROM table1 ;

GO
```