SQL Assignments

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 FullName, FaxNumber, PhoneNumber from Application.People
   ```
2. If the customer's primary contact person has the same phone number as the customer's phone number, list the customer companies.
   ```
   with temp as(
     select a.CustomerCategoryID
     from Sales.Customers a
     inner join Sales.Customers b
     on a.CustomerID = b.PrimaryContactPersonID
     and a.PhoneNumber = b.PhoneNumber
   )

   select c.CustomerCategoryName
   from Sales.CustomerCategories c
   inner join temp t
   on c.CustomerCategoryID = t.CustomerCategoryID;
   ```

3. List of customers to whom we made a sale prior to 2016 but no sale since 2016-01-01.
   ```
   with temp as (select distinct CustomerID
   from Sales.CustomerTransactions
   where CustomerID not in (select distinct CustomerID from Sales.CustomerTransactions
   where TransactionDate< '2016-01-01'))

   select t.CustomerID, CustomerName
   from temp t
   inner join Sales.Customers a
   on t.CustomerID = a.CustomerID;
   ```

4. List of Stock Items and total quantity for each stock item in Purchase Orders in Year 2013.
   ```
   with temp as(select StockItemID, count(*) as cnt from Warehouse.StockItemTransactions
   group by StockItemID)
   select StockItemName, cnt*QuantityPerOuter as count_of_quantity from temp t inner
   join Warehouse.StockItems s on t.StockItemID= s.StockItemID;
   ```

5. List of stock items that have at least 10 characters in description.
   ```
   select StockItemName from Warehouse.StockItems where MarketingComments is not
   null and len(MarketingComments)>=10;
   ```

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

```
select distinct a.StockItemID, e.StockItemName from
WideWorldImporters.Warehouse.StockItemTransactions a
inner join WideWorldImporters.Sales.Customers b on a.CustomerID = b.CustomerID
inner join WideWorldImporters.Application.Cities c on b.DeliveryCityID= c.CityID
inner join WideWorldImporters.Application.StateProvinces d on c.StateProvinceID =
d.StateProvinceID
inner join WideWorldImporters.Warehouse.StockItems e on a.StockItemID =
e.StockItemID
inner join WideWorldImporters.Sales.Orders f on f.CustomerID=a.CustomerID
where year(f.OrderDate)='2014' and d.StateProvinceName not in ('Alabama', 'Georgia');
```

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

```
with temp as(
   select a.InvoiceID, b.CustomerID, DATEDIFF(day, b.OrderDate,
cast(a.ConfirmedDeliveryTime as date)) as diff_date
         from WideWorldImporters.Sales.Invoices a
         inner join WideWorldImporters.Sales.Orders b on a.OrderID = b.OrderID
)
select c.StateProvinceName, avg(diff_date) as avg_diff_date
from temp t left join WideWorldImporters.Sales.Customers a on t.CustomerID =
a.CustomerID
left join WideWorldImporters.Application.Cities b on a.DeliveryCityID= b.CityID
left join WideWorldImporters.Application.StateProvinces c on b.StateProvinceID =
c.StateProvinceID
group by c.StateProvinceName;
```

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

```
with temp as(
   select a.InvoiceID, b.CustomerID,month(b.OrderDate) as mth, DATEDIFF(day,
b.OrderDate, cast(a.ConfirmedDeliveryTime as date)) as diff_date
         from WideWorldImporters.Sales.Invoices a
         inner join WideWorldImporters.Sales.Orders b on a.OrderID = b.OrderID
)
select c.StateProvinceName, mth, avg(diff_date) as avg_diff_date
from temp t left join WideWorldImporters.Sales.Customers a on t.CustomerID =
a.CustomerID
left join WideWorldImporters.Application.Cities b on a.DeliveryCityID= b.CityID
left join WideWorldImporters.Application.StateProvinces c on b.StateProvinceID =
c.StateProvinceID
group by c.StateProvinceName, mth;
```

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

```
with temp1 as(

select b.StockItemID, sum(b.Quantity) as cnt_sale

from WideWorldImporters.Warehouse.StockItems a

right join WideWorldImporters.Sales.OrderLines b on a.StockItemID = b.StockItemID

left join WideWorldImporters.Sales.Orders c on b.OrderID= c.OrderID

where year(c.OrderDate) = '2015'

group by b.StockItemID),

temp2 as(

select e.StockItemID, sum(e.ReceivedOuters) as cnt_purchase

from WideWorldImporters.Warehouse.StockItems d

right join WideWorldImporters.Purchasing.PurchaseOrderLines e on d.StockItemID =
e.StockItemID

left join WideWorldImporters.Purchasing.PurchaseOrders f on e.PurchaseOrderID =
f.PurchaseOrderID

where year(f.OrderDate) = '2015'

group by e.StockItemID

)

select e.StockItemID

from temp1 e

inner join temp2 f

on e.StockItemID= f.StockItemID

where cnt_purchase> cnt_sale;
```

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.

```
with temp as(

select d.CustomerName
```

from Sales.Orders a left join Sales.OrderLines b on a.OrderID = b.OrderID

left join Warehouse.StockItems c on b.StockItemID = c.StockItemID

left join Sales.Customers d on a.CustomerID = d.CustomerID

where year(a.OrderDate) = 2016 and c.StockItemName like '%mug%'

group by d.CustomerName having count(*) < 10)

select t.*, g.PhoneNumber, f.CustomerName as PrimaryName

from temp t inner join Sales.Customers g on t.CustomerName = g.CustomerName

inner join Sales.Customers f on g.PrimaryContactPersonID = f.CustomerID;

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

    select distinct CityName from Application.Cities where cast(ValidFrom as date) > '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 a.StockItemName, e.CityName,f.StateProvinceName, g.CountryName, d.CustomerName, d.PrimaryContactPersonID,d.PhoneNumber, b.Quantity

    From Warehouse.StockItems a inner join Sales.OrderLines b on a.StockItemID = b.StockItemID

    inner join Sales.Orders c on b.OrderID = C.OrderID

    inner join Sales.Customers d on c.CustomerID = d.CustomerID

    inner join Application.Cities e on d.DeliveryCityID = e.CityID

    inner join Application.StateProvinces f on e.StateProvinceID = f.StateProvinceID

    inner join Application.Countries g on f.CountryID = g.CountryID

13. List of stock item groups and total quantity purchased, total quantity sold, and the remaining stock quantity (quantity purchased – quantity sold)
    select *, QuantityPurchased - QuantitySold as diff from(
    select b.StockItemStockGroupID, sum(c.OrderedOuters) as QuantityPurchased, sum(a.Quantity) as QuantitySold
    from Sales.OrderLines a inner join Warehouse.StockItemStockGroups b on a.StockItemID = b.StockItemID
    inner join Purchasing.PurchaseOrderLines c on c.StockItemID = b.StockItemID

Group by b.StockItemStockGroupID) a

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 temp as(
    select CityName, StockItemName from(
    select a.CityName, e.StockItemName, rank() over(partition by a.CityName order by COUNT(c.OrderID) desc) as cnt
    from Application.Cities a left join Sales.Customers b on a.CityID =b.DeliveryMethodID
    inner join Sales.Orders c on b.CustomerID = C.CustomerID
    inner JOIN Sales.OrderLines d on c.OrderID = d.OrderID
    inner JOIN Warehouse.StockItems e on e.StockItemID= d.StockItemID
    where year(c.OrderDate) = '2016'
    Group by a.CityName, e.StockItemName) a
    where cnt = 1)

    select b.CityName, coalesce(StockItemName, 'No Sales') from temp t right join Application.Cities b on b.CityName = t.CityName;

15. List any orders that had more than one delivery attempt (located in invoice table).
    select CustomerID FROM Sales.Invoices group by CustomerID having count(*)>1;
16. List all stock items that are manufactured in China. (Country of Manufacture)
    select [a].[StockItemName]
    from [Warehouse].[StockItems] a
    where JSON_VALUE(a.CustomFields, '$.CountryOfManufacture') = 'China';

17. Total quantity of stock items sold in 2015, group by country of manufacturing.
    select JSON_VALUE(a.CustomFields, '$.CountryOfManufacture') as country, sum([b].[Quantity]) as SumQuantity
    from [Warehouse].[StockItems] a left join [Sales].[OrderLines] b on [a].[StockItemID] = [b].[StockItemID]
    LEFT JOIN [Sales].[Orders] c on [b].[OrderID] = [c].[OrderID]
    where [c].[OrderDate] = '2015'
    group by JSON_VALUE(a.CustomFields, '$.CountryOfManufacture') ;

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]
    Select c.StockGroupName, sum(case when year(d.OrderDate) = '2013' then a.Quantity else 0 end) as cnt2013,
    sum(case when year(d.OrderDate) = '2014' then a.Quantity else 0 end) as cnt2014,
    sum(case when year(d.OrderDate) = '2015' then a.Quantity else 0 end) as cnt2015,
    sum(case when year(d.OrderDate) = '2016' then a.Quantity else 0 end) as cnt2016,
    sum(case when year(d.OrderDate) = '2017' then a.Quantity else 0 end) as cnt2017
    from Sales.OrderLines a inner join Warehouse.StockItemStockGroups b on a.StockItemID = b.StockItemID

```
inner join Warehouse.StockGroups c on b.StockGroupID = c.StockGroupID
inner join Sales.Orders d on a.OrderID = d.OrderID
group by c.StockGroupName
```

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]

```
SELECT
    y as year,[T-Shirts],[USB Novelties],[Packaging Materials],[Clothing],[Novelty
Items],[Furry Footwear],[Mugs],[Computing Novelties],[Toys]
  FROM
    (select c.StockGroupName, a.Quantity,year(d.OrderDate) as y
            from Sales.OrderLines a inner join Warehouse.StockItemStockGroups b
on a.StockItemID = b.StockItemID
    inner join Warehouse.StockGroups c on b.StockGroupID = c.StockGroupID
    inner join Sales.Orders d on a.OrderID = d.OrderID)
 t
PIVOT(
  SUM(Quantity)
  FOR StockGroupName IN (
    [T-Shirts],
[USB Novelties],
[Packaging Materials],
[Clothing],
[Novelty Items],
[Furry Footwear],
[Mugs],
[Computing Novelties],
[Toys])
) AS pivot_table
order by y;
```

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 dbo.totaloforder (@IDinput int)
RETURNS float As
Begin
        DECLARE @OrderTotal float;
        SELECT @OrderTotal=sum(ol.Quantity*ol.UnitPrice*(1+ol.TaxRate/100)) from
Sales.OrderLines ol
        where ol.OrderID= @IDinput group by ol.OrderID

        IF (@OrderTotal IS NULL)
                SET @OrderTotal = 0
```

```
            RETURN @OrderTotal
End;

select i.OrderID, dbo.totaloforder(i.OrderID) OrderTotal from sales.Invoices i;
```

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 SCHEMA ods;
GO

CREATE TABLE ods.orders (
OrderID int,
OrderDate date,
OrderTotal decimal(18, 2)           ,
CustomerID int,
);




Alter PROCEDURE inputdate (@inputdate DATE)
AS
BEGIN TRY
            BEGIN TRANSACTION;
                        IF EXISTS (SELECT * from ods.orders where OrderDate= @inputdate)
                                    RAISERROR('Date Already Exists', 16, 1)
                        ELSE
                                    with Orderinfo(OrderID, OrderTotal)
                                    AS(SELECT ol.OrderID,
sum(ol.Quantity*ol.UnitPrice*(1+ol.TaxRate/100)) OrderTotal from Sales.OrderLines ol
group by ol.OrderID)

                                    INSERT into ods.orders
(OrderID,OrderDate,OrderTotal,CustomerID )select
oi.OrderID,o.OrderDate,oi.OrderTotal,o.CustomerID
                                    from Orderinfo oi join sales.Orders o on oi.OrderID = o.OrderID
                                    where o.OrderDate= @inputdate
                                    COMMIT
END TRY
```

```
BEGIN CATCH
        IF @@TRANCOUNT > 0
                ROLLBACK
        DECLARE @ErrorMessage NVARCHAR(4000), @ErrorSeverity INT;
        SELECT @ErrorMessage = ERROR_MESSAGE(),@ErrorSeverity =
ERROR_SEVERITY();
        RAISERROR(@ErrorMessage, @ErrorSeverity, 1);
END CATCH;

EXEC inputdate '2013-01-10';EXEC inputdate '2013-01-11';EXEC inputdate '2013-01-12';
EXEC inputdate '2013-01-13';EXEC inputdate '2013-01-14';
select * from ods.orders
```

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,
   StockItemName nvarchar(100),
        SupplierID int,ColorID int,
        UnitPackageID int,OuterPackageID int,
        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),
        CountryOfManufacture nvarchar(50),[Range] nvarchar(50),Shelflife nvarchar(50)
);

INSERT Into ods.StockItem([StockItemID], [StockItemName] ,[SupplierID] ,[ColorID]
,[UnitPackageID] ,[OuterPackageID] ,[Brand] ,[Size] ,
[LeadTimeDays] ,[QuantityPerOuter] ,[IsChillerStock] ,[Barcode] ,[TaxRate]
,[UnitPrice],[RecommendedRetailPrice] ,
[TypicalWeightPerUnit] ,[MarketingComments],[InternalComments],
[CountryOfManufacture], [Range], [Shelflife])

select l1.[StockItemID], l1.[StockItemName] ,l1.[SupplierID] ,l1.[ColorID]
,l1.[UnitPackageID] ,l1.[OuterPackageID],l1.[Brand] ,l1.[Size] ,
l1.[LeadTimeDays] ,l1.[QuantityPerOuter] ,l1.[IsChillerStock] ,l1.[Barcode] ,l1.[TaxRate]
,l1.[UnitPrice],l1.[RecommendedRetailPrice] ,
l1.[TypicalWeightPerUnit] ,l1.[MarketingComments],l1.[InternalComments],l1.[origin],
l1.[Range], l1.[Shelflife] from
(SELECT *, JSON_VALUE(si.CustomFields,'$.CountryOfManufacture') AS origin,
JSON_VALUE(si.CustomFields,'$.Range') AS [Range],
JSON_VALUE(si.CustomFields,'$.ShelfLife') AS ShelfLife
FROM Warehouse.StockItems si) l1

select *from ods.StockItem
```

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.

```
select * from ods.orders
Alter PROCEDURE inputdate_load7days (@inputdate DATE)
AS

DELETE FROM ods.orders WHERE OrderDate<@inputdate;

with Orderinfo(OrderID, OrderTotal)
AS(SELECT ol.OrderID,  sum(ol.Quantity*ol.UnitPrice*(1+ol.TaxRate/100)) OrderTotal
from Sales.OrderLines ol group by ol.OrderID)

INSERT into ods.orders (OrderID,OrderDate,OrderTotal,CustomerID )select
oi.OrderID,o.OrderDate,oi.OrderTotal,o.CustomerID
from Orderinfo oi join sales.Orders o on oi.OrderID = o.OrderID
where o.OrderDate between @inputdate AND DATEADD(day, 7,@inputdate)

EXEC inputdate_load7days '2013-01-05';
select * from ods.orders
```

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.

```
        (SELECT * FROM OPENJSON(@json, '$')
        WITH  (
                [StockItemName]    nvarchar(50)
        '$.PurchaseOrders[0].StockItemName',
                [Supplier]  int     '$.PurchaseOrders[0].Supplier',
                [UnitPackageId]      int      '$.PurchaseOrders[0].UnitPackageId',
                [OuterPackageId]     int  '$.PurchaseOrders[0].OuterPackageId[0]',
                      [Brand]      nvarchar(50)  '$.PurchaseOrders[0].Brand',
                      [LeadTimeDays]      int  '$.PurchaseOrders[0].LeadTimeDays',
              [QuantityPerOuter]      int
'$.PurchaseOrders[0].QuantityPerOuter',
              [TaxRate]      int  '$.PurchaseOrders[0].TaxRate',
              [UnitPrice]      DEC(10,2)  '$.PurchaseOrders[0].UnitPrice',
              [RecommendedRetailPrice]      DEC(10,2)
'$.PurchaseOrders[0].RecommendedRetailPrice',
              [TypicalWeightPerUnit]      DEC(10,2)
'$.PurchaseOrders[0].TypicalWeightPerUnit',
              [CountryOfManufacture]      nvarchar(50)
'$.PurchaseOrders[0].CountryOfManufacture',
              [Range]      nvarchar(50)  '$.PurchaseOrders[0].Range',
              [OrderDate]      datetime '$.PurchaseOrders[0].OrderDate',
              [DeliveryMethod]      nvarchar(10)
'$.PurchaseOrders[0].DeliveryMethod',
              [ExpectedDeliveryDate]      datetime
'$.PurchaseOrders[0].ExpectedDeliveryDate',
              [SupplierReference]      nvarchar(MAX)
'$.PurchaseOrders[0].SupplierReference'
    )
) UNION
(
SELECT * FROM OPENJSON(@json, '$')
WITH  (
        [StockItemName]    nvarchar(50)
'$.PurchaseOrders[0].StockItemName',
        [Supplier]  int     '$.PurchaseOrders[0].Supplier',
        [UnitPackageId]       int       '$.PurchaseOrders[0].UnitPackageId',
        [OuterPackageId]      int  '$.PurchaseOrders[0].OuterPackageId[1]',
              [Brand]      nvarchar(50)  '$.PurchaseOrders[0].Brand',
              [LeadTimeDays]      int  '$.PurchaseOrders[0].LeadTimeDays',
              [QuantityPerOuter]      int
'$.PurchaseOrders[0].QuantityPerOuter',
              [TaxRate]      int  '$.PurchaseOrders[0].TaxRate',
              [UnitPrice]      DEC(10,2)  '$.PurchaseOrders[0].UnitPrice',
              [RecommendedRetailPrice]      DEC(10,2)
'$.PurchaseOrders[0].RecommendedRetailPrice',
              [TypicalWeightPerUnit]      DEC(10,2)
'$.PurchaseOrders[0].TypicalWeightPerUnit',
              [CountryOfManufacture]      nvarchar(50)
'$.PurchaseOrders[0].CountryOfManufacture',
              [Range]      nvarchar(50)  '$.PurchaseOrders[0].Range',
              [OrderDate]      datetime '$.PurchaseOrders[0].OrderDate',
              [DeliveryMethod]      nvarchar(10)
'$.PurchaseOrders[0].DeliveryMethod',
              [ExpectedDeliveryDate]      datetime
'$.PurchaseOrders[0].ExpectedDeliveryDate',
              [SupplierReference]      nvarchar(MAX)
'$.PurchaseOrders[0].SupplierReference'
    )
) UNION

(
SELECT * FROM OPENJSON(@json, '$')
WITH  (
        [StockItemName]    nvarchar(50)
'$.PurchaseOrders[1].StockItemName',
        [Supplier]  int     '$.PurchaseOrders[1].Supplier',
        [UnitPackageId]       int      '$.PurchaseOrders[1].UnitPackageId',
        [OuterPackageId]      int  '$.PurchaseOrders[1].OuterPackageId',
              [Brand]      nvarchar(50)  '$.PurchaseOrders[1].Brand',
              [LeadTimeDays]      int  '$.PurchaseOrders[1].LeadTimeDays',
              [QuantityPerOuter]      int
'$.PurchaseOrders[1].QuantityPerOuter',
              [TaxRate]      int  '$.PurchaseOrders[1].TaxRate',
              [UnitPrice]      DEC(10,2)  '$.PurchaseOrders[1].UnitPrice',
              [RecommendedRetailPrice]      DEC(10,2)
'$.PurchaseOrders[1].RecommendedRetailPrice',
              [TypicalWeightPerUnit]      DEC(10,2)
'$.PurchaseOrders[1].TypicalWeightPerUnit',
              [CountryOfManufacture]      nvarchar(50)
'$.PurchaseOrders[1].CountryOfManufacture',
              [Range]      nvarchar(50)  '$.PurchaseOrders[1].Range',
              [OrderDate]      datetime '$.PurchaseOrders[1].OrderDate',
              [DeliveryMethod]      nvarchar(10)
'$.PurchaseOrders[1].DeliveryMethod',
              [ExpectedDeliveryDate]      datetime
'$.PurchaseOrders[1].ExpectedDeliveryDate',
              [SupplierReference]      nvarchar(MAX)
'$.PurchaseOrders[1].SupplierReference'
    )
)
```

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

select * from(
SELECT l1.StockGroupName,L1.stockyear,ABS(SUM(L1.Quantity)) Quant FROM
(
SELECT sg.StockGroupName, year(st.TransactionOccurredWhen) as stockyear,st.Quantity
FROM Warehouse.StockGroups sg
join Warehouse.StockItemStockGroups sisg on sg.StockGroupID = sisg.StockGroupID
join Warehouse.StockItems si on si.StockItemID = sisg.StockItemID
join Warehouse.StockItemTransactions st on st.StockItemID = si.StockItemID and ST.Quantity<0
) L1 GROUP BY L1.stockyear, L1.StockGroupName) l2

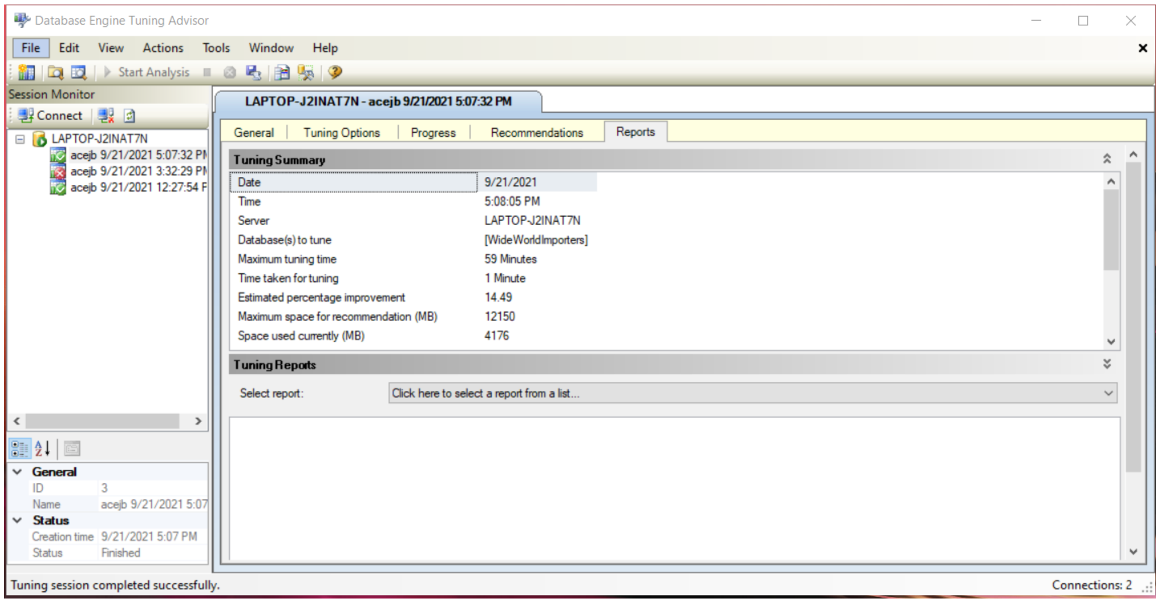
pivot
(
sum([Quant])
for StockGroupName in([Novelty Items],[Clothing],[Mugs],[T-Shirts],[Computing Novelties],[USB Novelties],
[Furry Footwear],[Toys],[Packaging Materials])
)As pvi

select * from stockbygroup order by stockyear FOR JSON Path

26. Revisit your answer in (19). Convert the result into an XML string and save it to the server using TSQL FOR XML PATH.
    select * from stockbygroup order by stockyear FOR XML AUTO
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.

```
DROP TABLE IF EXISTS NewTable;
CREATE TABLE NewTable (
id INT,
date datetime,
value nvarchar(MAX)
);

CREATE OR ALTER PROC GenJSON(
       @InputDate datetime,
       @CustomerId INT = 1,
       @json nvarchar(MAX) OUTPUT
)AS
BEGIN
SET @json = (
SELECT DISTINCT SI.CustomerID as [customer.id],(SELECT SI.BillToCustomerID,
SI.DeliveryMethodID,
SIL.InvoiceLineID, SIL.Description  FROM  Sales.Invoices SI JOIN
Sales.InvoiceLines SIL ON
SI.InvoiceID = SIL.InvoiceID WHERE SI.CustomerID = 1 AND
CONVERT(date,SI.ConfirmedDeliveryTime) = @InputDate FOR JSON PATH) as
[customer.deliveries]
FROM Sales.Invoices SI WHERE SI.CustomerID=1 FOR JSON PATH);
END;

CREATE OR ALTER PROC GenerateAnswer AS
BEGIN
DECLARE @dTime TABLE(
       deliveredDates datetime,
       numbers INT);
DECLARE @json nvarchar(MAX);
DECLARE @totalRow INT = 0;
DECLARE @rowCounter INT = 0;
DECLARE @curTime datetime;
INSERT INTO @dTime SELECT tt.deliveredDates,tt.numbers FROM (SELECT
DISTINCT CONVERT(date, SI.ConfirmedDeliveryTime)
as deliveredDates,
row_number() over(ORDER BY CONVERT(date, SI.ConfirmedDeliveryTime)) AS
numbers FROM Sales.Invoices SI
WHERE SI.CustomerID=1) tt
SET @totalRow =  @@ROWCOUNT;

WHILE @rowCounter < @totalRow
       BEGIN
       SET @curTime = (SELECT CONVERT(date,deliveredDates) FROM @dTime
       WHERE numbers = @rowCounter+1);
       EXEC GenJSON @InputDate = @curTime,@json=@json OUTPUT;
       INSERT INTO NewTable (id, date, value) VALUES(@rowCounter+1,
@curTime, @json);
       SET @rowCounter = @rowCounter + 1;
       END;
END;
EXEC GenerateAnswer;
SELECT * FROM NewTable;
```

28. Write a short essay talking about your understanding of transactions, locks and isolation levels.
    Transaction is a unit of work that is against the database. For each transaction, if multiple server users read or update particular rows , the rows will be locked only allowing one user to read the rows, which is called isolation. There are five types of isolation levels: read uncommitted, read committed, repeatable read, serializable and snapshot isolation.
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.
    Performance tuning can't be told without performance monitoring, which includes extended events, DMV, logs and execution plan. By using Tuning Advisor, you can get a recommendation of how much percent can be improved using the recommended index key. The execution plan will also show what has been changed to index, such as changing from clustered key to nonclustered key.



Database Engine Tuning Advisor
File Edit View Actions Tools Window Help
Start Analysis
Session Monitor
Connect
LAPTOP-J2INAT7N
acejb 9/21/2021 5:07:32 PM
acejb 9/21/2021 3:32:29 PM
acejb 9/21/2021 12:27:54 PM
General Tuning Options Progress Recommendations Reports

LAPTOP-J2INAT7N - acejb 9/21/2021 5:07:32 PM

Tuning Summary

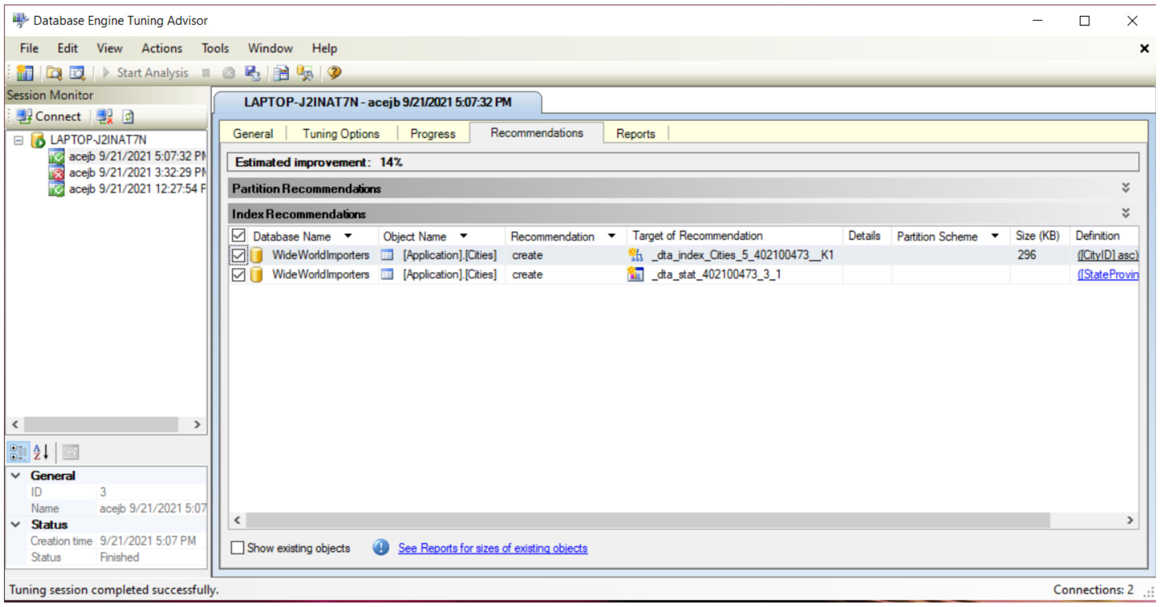
Date	9/21/2021
Time	5:08:05 PM
Server	LAPTOP-J2INAT7N
Database(s) to tune	[WideWorldImporters]
Maximum tuning time	59 Minutes
Time taken for tuning	1 Minute
Estimated percentage improvement	14.49
Maximum space for recommendation (MB)	12150
Space used currently (MB)	4176

Tuning Reports

Select report: Click here to select a report from a list...

General	
ID	3
Name	acejb 9/21/2021 5:07
Status	
Creation time	9/21/2021 5:07 PM
Status	Finished

Tuning session completed successfully. Connections: 2



Database Engine Tuning Advisor
File Edit View Actions Tools Window Help
Start Analysis
Session Monitor
Connect
LAPTOP-J2INAT7N
acejb 9/21/2021 5:07:32 PM
acejb 9/21/2021 3:32:29 PM
acejb 9/21/2021 12:27:54 PM
General Tuning Options Progress Recommendations Reports

LAPTOP-J2INAT7N - acejb 9/21/2021 5:07:32 PM

Estimated improvement: 14%

Partition Recommendations

Index Recommendations

☑	Database Name	Object Name	Recommendation	Target of Recommendation	Details	Partition Scheme	Size (KB)	Definition
☑	WideWorldImporters	[Application].[Cities]	create	_dta_index_Cities_5_402100473__K1			296	([CityID] asc)
☑	WideWorldImporters	[Application].[Cities]	create	_dta_stat_402100473_3_1				([StateProvin

☐ Show existing objects   See Reports for sizes of existing objects

General	
ID	3
Name	acejb 9/21/2021 5:07
Status	
Creation time	9/21/2021 5:07 PM
Status	Finished

Tuning session completed successfully. Connections: 2

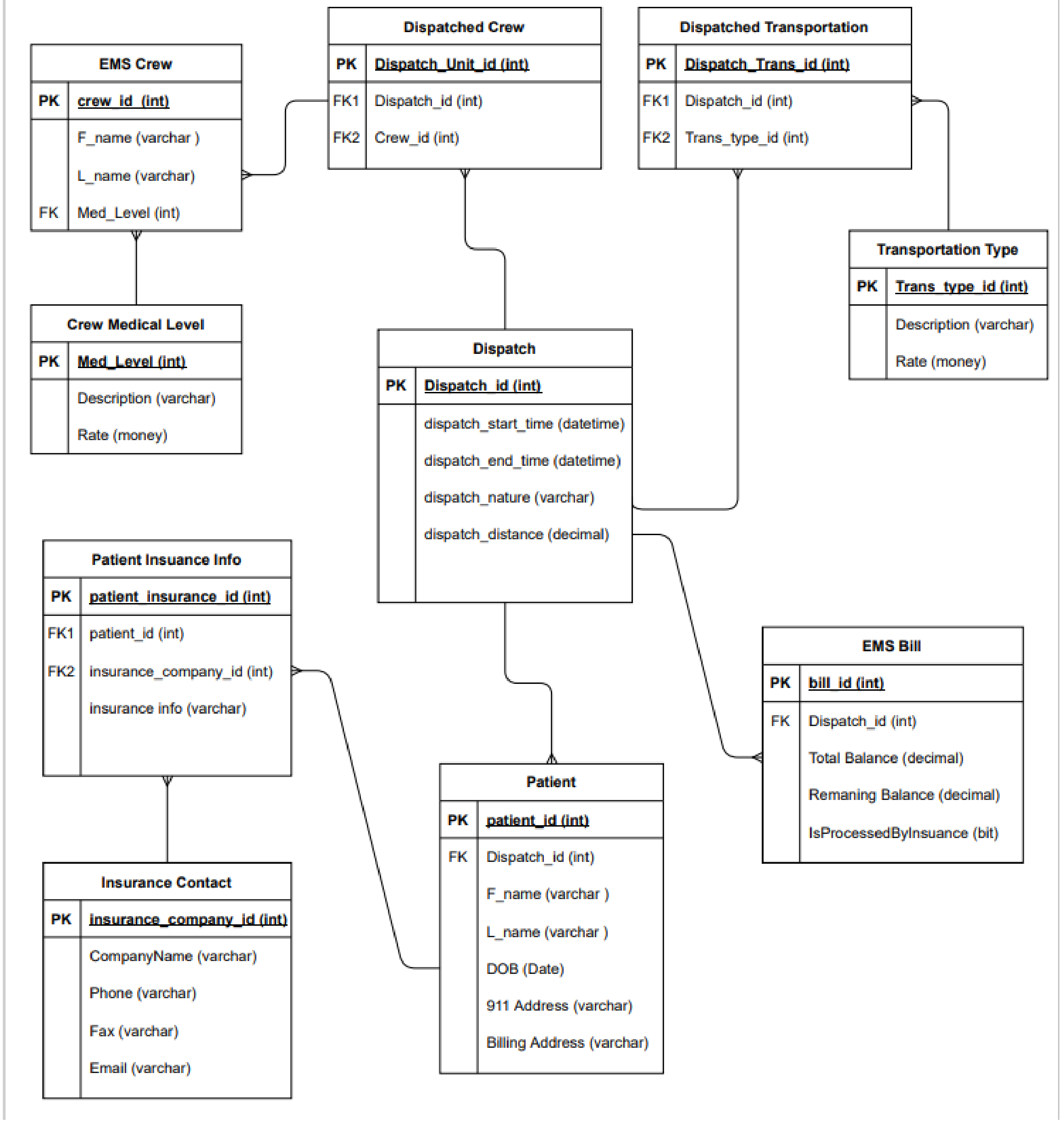
Assignments 30 - 32 are group assignments.

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.

```
Use AdventureWorks2019

--add supplier category
INSERT INTO WideWorldImporters.Purchasing.SupplierCategories
(SupplierCategoryName, LastEditedBy)
values('Bike Supplier',1)

--add vendor to supplier
INSERT INTO WideWorldImporters.Purchasing.Suppliers (SupplierName,
SupplierCategoryID, PrimaryContactPersonID,
AlternateContactPersonID, DeliveryCityID, PostalCityID, PaymentDays, PhoneNumber,
FaxNumber,WebsiteURL, [DeliveryAddressLine1],
[DeliveryPostalCode],[PostalAddressLine1],[PostalPostalCode],[LastEditedBy])
select [Name], 11, 21, 22,38171,38171,14,'','','','','','','',1
FROM AdventureWorks2019.Purchasing.Vendor
where [Name] COLLATE Latin1_General_100_CI_AI Not IN (select s.SupplierName from
WideWorldImporters.Purchasing.Suppliers s)

--combine product category    Warehouse.StockGroups
Insert into WideWorldImporters.Warehouse.StockGroups (StockGroupName,
LastEditedBy)
select [Name],1 from Production.ProductCategory
where [Name] COLLATE Latin1_General_100_CI_AI Not IN (select sg.StockGroupName
from WideWorldImporters.Warehouse.StockGroups sg)

--insert item    Warehouse.StockItems
Insert into WideWorldImporters.Warehouse.StockItems ([StockItemName],
[SupplierID],[ColorID],[UnitPackageID],
[OuterPackageID],[Size],[LeadTimeDays],[QuantityPerOuter],[IsChillerStock],[TaxRate],[U
nitPrice],[RecommendedRetailPrice],
[TypicalWeightPerUnit],[LastEditedBy]
)

select concat (pro.[Name] COLLATE Latin1_General_100_CI_AI, ' (', s.SupplierName , ')'),
s.SupplierID,
ColorID,6 [UnitPackageID],7 [OuterPackageID], pro.Size,pv.AverageLeadTime,
1 [QuantityPerOuter], 0 [IsChillerStock],0 [TaxRate],
pro.StandardCost, pro.ListPrice, 0, 1 from Production.Product pro
left join WideWorldImporters.Warehouse.Colors color on color.ColorName = pro.Color
COLLATE Latin1_General_100_CI_AI
join Purchasing.ProductVendor pv on pv.ProductID = pro.ProductID
join Purchasing.Vendor v on pv.BusinessEntityID = v.BusinessEntityID
join WideWorldImporters.Purchasing.Suppliers s on s.SupplierName = v.[Name] COLLATE
Latin1_General_100_CI_AI
where pro.[Name] COLLATE Latin1_General_100_CI_AI Not IN (select si.StockItemName
from WideWorldImporters.Warehouse.StockItems si)


--assign product to category Warehouse.StockItemStockGroups
Insert into WideWorldImporters.Warehouse.StockItemStockGroups
([StockItemID],[StockGroupID],[LastEditedBy])
select s.StockItemID,sg.StockGroupID,1
from Production.Product pro
join Production.ProductModel pmodel on pmodel.ProductModelID =
pro.ProductModelID
join Purchasing.ProductVendor pv on pv.ProductID = pro.ProductID
join Purchasing.Vendor v on pv.BusinessEntityID = v.BusinessEntityID
join Production.ProductSubcategory pscat on pscat.ProductSubcategoryID =
pro.ProductSubcategoryID
join Production.ProductCategory pcat on pcat.ProductCategoryID =
pscat.ProductCategoryID
join WideWorldImporters.Warehouse.StockItems s on s.StockItemName = concat
(pro.[Name] COLLATE Latin1_General_100_CI_AI, ' (', v.[Name], ')')
join WideWorldImporters.Warehouse.StockGroups sg on sg.StockGroupName =
pcat.[Name] COLLATE Latin1_General_100_CI_AI

---for people table
--merge to application.people
Insert into WideWorldImporters.[Application].People([FullName],[PreferredName],
[IsPermittedToLogon],[IsExternalLogonProvider],[HashedPassword],[IsSystemUser],
[IsEmployee],[IsSalesperson],[PhoneNumber],[EmailAddress],[CustomFields],[LastEdited
By])

select CONCAT(p.FirstName, ' ', MiddleName,' ',p.LastName ) FullName,
p.FirstName,0, 0, convert(varbinary(max),pwd.PasswordHash), 0,
case when p.PersonType='EM' THEN 1 ELSE 0 END as IsEmployee,
case when p.PersonType= 'SP' THEN 1 ELSE 0 END as IsSalesperson,
phone.PhoneNumber,e.EmailAddress,
(select emp.JobTitle, emp.HireDate FOR JSON PATH)  CustomFields,1
from Person.EmailAddress e
join Person.[Password] pwd on e.BusinessEntityID = pwd.BusinessEntityID
join Person.PersonPhone phone on e.BusinessEntityID = phone.BusinessEntityID
join Person.Person p on e.BusinessEntityID = p.BusinessEntityID
join HumanResources.Employee emp on emp.BusinessEntityID = p.BusinessEntityID
```

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.

```
DROP TYPE [dbo].[MemoryType]

Delete from [Integration].Order_Staging
CREATE TYPE [dbo].[MemoryType]
  AS TABLE
  (
        [Order Staging Key] [bigint] PRIMARY KEY NONCLUSTERED,
        [City Key] [int] NULL,
        [Customer Key] [int] NULL,
        [Stock Item Key] [int] NULL,
        [Order Date Key] [date] NULL,
        [Picked Date Key] [date] NULL,
        [Salesperson Key] [int] NULL,
        [Picker Key] [int] NULL,
        [WWI Order ID] [int] NULL,
        [WWI Backorder ID] [int] NULL,
        [Description] [nvarchar](100) COLLATE Latin1_General_100_CI_AS NULL,
        [Package] [nvarchar](50) COLLATE Latin1_General_100_CI_AS NULL,
        [Quantity] [int] NULL,
        [Unit Price] [decimal](18, 2) NULL,
        [Tax Rate] [decimal](18, 3) NULL,
        [Total Excluding Tax] [decimal](18, 2) NULL,
        [Tax Amount] [decimal](18, 2) NULL,
        [Total Including Tax] [decimal](18, 2) NULL,
        [Lineage Key] [int] NULL,
        [WWI City ID] [int] NULL,
        [WWI Customer ID] [int] NULL,
        [WWI Stock Item ID] [int] NULL,
        [WWI Salesperson ID] [int] NULL,
        [WWI Picker ID] [int] NULL,
        [Last Modified When] [datetime2](7) NULL
  )
  WITH
    (MEMORY_OPTIMIZED = ON);
```

```
GO

DECLARE @InMem dbo.MemoryType;
INSERT into @InMem ([Order Staging Key],[WWI City ID],[WWI Customer
ID],[WWI Stock Item ID],[Order Date Key],[Picked Date Key]
,[WWI Salesperson ID],[WWI Picker ID],[WWI Order ID],[Description]
,[Package],[Quantity],[Unit Price],[Tax Rate],[Total Excluding Tax]
,[Tax Amount],[Total Including Tax])

select ROW_NUMBER() OVER(ORDER BY o.OrderID ASC), ci.CityID, o.CustomerID,
si.StockItemID,  o.OrderDate, CONVERT(date, ol.PickingCompletedWhen)
PickupDate,
o.SalespersonPersonID, o.PickedByPersonID,
o.OrderID, ol.[Description], 'Each' Package, ol.Quantity,ol.UnitPrice,ol.TaxRate,
(ol.Quantity*ol.UnitPrice) TotalExcludingTax,
(ol.Quantity*ol.UnitPrice)*(ol.TaxRate/100) TaxAmount,
(ol.Quantity*ol.UnitPrice)*(1+ol.TaxRate/100)TotalIncludeTax
from WideWorldImporters.Sales.Orders o
join WideWorldImporters.Sales.OrderLines ol on  o.OrderID = ol.OrderID
join WideWorldImporters.Warehouse.StockItems si on si.StockItemID =
ol.StockItemID
JOIN WideWorldImporters.Sales.Customers cus ON cus.CustomerID =
o.CustomerID
JOIN WideWorldImporters.[Application].Cities ci ON ci.CityID =
cus.DeliveryCityID

Insert Into [WideWorldImportersDW].[Integration].Order_Staging
([WWI City ID],[WWI Customer ID],[WWI Stock Item ID],[Order Date Key],[Picked
Date Key]
,[WWI Salesperson ID],[WWI Picker ID],[WWI Order ID],[Description]
,[Package],[Quantity],[Unit Price],[Tax Rate],[Total Excluding Tax]
,[Tax Amount],[Total Including Tax])
select [WWI City ID],[WWI Customer ID],[WWI Stock Item ID],[Order Date
Key],[Picked Date Key]
,[WWI Salesperson ID],[WWI Picker ID],[WWI Order ID],[Description]
,[Package],[Quantity],[Unit Price],[Tax Rate],[Total Excluding Tax]
,[Tax Amount],[Total Including Tax] from @InMem

--select *from [WideWorldImportersDW].[Integration].Order_Staging
--delete from [WideWorldImportersDW].[Integration].Order_Staging
-----------------------END of Intergration Now all in DW
table----------------------------------


--update intergation to match the key
```

```
--match city key
UPDATE [WideWorldImportersDW].[Integration].Order_Staging
SET [WideWorldImportersDW].[Integration].Order_Staging.[City Key] = c.[City
Key]
FROM [WideWorldImportersDW].[Integration].Order_Staging os
INNER JOIN [WideWorldImportersDW].Dimension.City C
ON c.[WWI City ID] = os.[WWI City ID];

--match stock item key
UPDATE [WideWorldImportersDW].[Integration].Order_Staging
SET [WideWorldImportersDW].[Integration].Order_Staging.[Stock Item Key] =
si.[Stock Item Key]
FROM [WideWorldImportersDW].[Integration].Order_Staging os
INNER JOIN [WideWorldImportersDW].Dimension.[Stock Item] si
ON si.[WWI Stock Item ID] = os.[WWI Stock Item ID];

--match sales person
UPDATE [WideWorldImportersDW].[Integration].Order_Staging
SET [WideWorldImportersDW].[Integration].Order_Staging.[Salesperson Key] =
e.[Employee Key]
FROM [WideWorldImportersDW].[Integration].Order_Staging os
INNER JOIN [WideWorldImportersDW].Dimension.Employee e
ON e.[WWI Employee ID] = os.[WWI Salesperson ID];

--match picker
UPDATE [WideWorldImportersDW].[Integration].Order_Staging
SET [WideWorldImportersDW].[Integration].Order_Staging.[Picker Key] =
e.[Employee Key]
FROM [WideWorldImportersDW].[Integration].Order_Staging os
INNER JOIN [WideWorldImportersDW].Dimension.Employee e
ON e.[WWI Employee ID] = os.[WWI Picker ID];
GO

--Insert into fact table
Insert Into WideWorldImportersDW.Fact.[Order]([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])

select [City Key],0 [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], 0 [Lineage Key]
from Integration.Order_Staging

--clear table
delete from [Integration].Order_Staging
```