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,
    p.FaxNumber,
    p.PhoneNumber,
    c.PhoneNumber AS CompanyPhoneNumber,
    c.FaxNumber AS CompanyFaxNumber
FROM
    Application.People AS p
LEFT JOIN
    Sales.Customers AS c
ON
    p.PersonId = c.PrimaryContactPersonID OR p.PersonID = c.AlternateContac
```

100 %

Results Messages

	FullName	FaxNumber	PhoneNumber	CompanyPhoneNumber	CompanyFaxNumber
1	Data Conversion Only	NULL	NULL	NULL	NULL
2	Kayla Woodcock	(415) 555-0103	(415) 555-0102	NULL	NULL
3	Hudson Onslow	(415) 555-0103	(415) 555-0102	NULL	NULL
4	Isabella Rupp	(415) 555-0103	(415) 555-0102	NULL	NULL
5	Eva Muirden	(415) 555-0103	(415) 555-0102	NULL	NULL

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

```
SELECT
    c.CustomerName
FROM
    Application.People AS p
INNER JOIN
    Sales.Customers AS c
ON
    p.PersonID = c.PrimaryContactPersonID
WHERE
    p.PhoneNumber = c.PhoneNumber;
```

100 %

Results Messages

	CustomerName
1	Tailspin Toys (Head Office)
2	Tailspin Toys (Sylvanite, MT)
3	Tailspin Toys (Peeples Valley, AZ)
4	Tailspin Toys (Medicine Lodge, KS)
5	Tailspin Toys (Gasport, NY)

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

```
SELECT
    o.CustomerID
FROM
    Sales.Orders AS o
WHERE
    o.OrderDate < '2016-01-01' AND o.CustomerID NOT IN (
    SELECT
        DISTINCT CustomerID
    FROM
        Sales.Orders
    WHERE
        Sales.Orders.OrderDate >= '2016-01-01');
```

100 %

Results Messages

CustomerID

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

```
SELECT
    s.StockItemName,
    COUNT(s.StockItemName) AS quantity
FROM
    Purchasing.PurchaseOrders AS p
LEFT JOIN
    Purchasing.PurchaseOrderLines AS pol
ON
    pol.PurchaseOrderID = p.PurchaseOrderID
LEFT JOIN
    Warehouse.StockItems_Archive AS s
ON
    pol.StockItemID = s.StockItemID
WHERE
    YEAR(P.OrderDate) ='2013'
GROUP BY
    s.StockItemName;
```

100 %

Results Messages

	StockItemName	quantity
1	Furry animal socks (Pink) XL	1
2	"The Gu" red shirt XML tag t-shirt (White) L	1
3	20 mm Anti static bubble wrap (Blue) 20m	2
4	Permanent marker red 5mm nib (Red) 5mm	2
5	Shipping carton (Brown) 457x457x457mm	2

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

```
SELECT
    s.StockItemName
FROM
    Warehouse.StockItems AS s
WHERE
    LEN(s.MarketingComments)>10;
```

100 %

Results | Messages

	StockItemName
1	USB missile launcher (Green)
2	USB rocket launcher (Gray)
3	Office cube periscope (Black)
4	Ride on toy sedan car (Black) 1/12 scale
5	Ride on toy sedan car (Red) 1/12 scale

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

```
SELECT DISTINCT si.StockItemName FROM Warehouse.StockItems AS si
LEFT JOIN Sales.OrderLines AS ol
ON OL.StockItemID = si.StockItemID
LEFT JOIN Sales.Orders AS o
ON o.OrderID = ol.OrderID
LEFT JOIN Sales.Customers AS c
ON c.CustomerID = o.CustomerID
LEFT JOIN Application.Cities
ON Cities.CityID = c.DeliveryCityID
LEFT JOIN Application.StateProvinces AS sp
ON Cities.StateProvinceID = sp.StateProvinceID
WHERE sp.StateProvinceName NOT IN ('Alabama', 'Georgia') AND YEAR(o.OrderDate) = 2014;
```

100 %

Results | Messages

	StockItemName
1	"The Gu" red shirt XML tag t-shirt (Black) 7XL
2	Halloween zombie mask (Light Brown) L
3	"The Gu" red shirt XML tag t-shirt (White) M
4	USB food flash drive - hamburger
5	USB rocket launcher (Gray)

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

```
SELECT
    sp.StateProvinceName,
    AVG(DATEDIFF(day, o.OrderDate, CONVERT(DATE,i.ConfirmedDeliveryTime))) AS AvgProcessDays
FROM Sales.Invoices AS i
LEFT JOIN Sales.Orders AS o
ON i.OrderID = o.OrderID
LEFT JOIN Sales.Customers AS c
ON c.CustomerID = o.CustomerID
LEFT JOIN Application.Cities
ON Cities.CityID = c.DeliveryCityID
LEFT JOIN Application.StateProvinces AS sp
ON Cities.StateProvinceID = sp.StateProvinceID
GROUP BY sp.StateProvinceName
ORDER BY sp.StateProvinceName;
```

100 %

Results Messages

	StateProvinceName	AvgProcessDays
1	Alabama	5
2	Alaska	5
3	Arizona	6
4	Arkansas	4
5	California	5

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

```
SELECT
    sp.StateProvinceName,
    MONTH(o.OrderDate) AS month,
    AVG(DATEDIFF(day, o.OrderDate, CONVERT(DATE,i.ConfirmedDeliveryTime))) AS AvgProcessDays
FROM Sales.Invoices AS i
LEFT JOIN Sales.Orders AS o
ON i.OrderID = o.OrderID
LEFT JOIN Sales.Customers AS c
ON c.CustomerID = o.CustomerID
LEFT JOIN Application.Cities
ON Cities.CityID = c.DeliveryCityID
LEFT JOIN Application.StateProvinces AS sp
ON Cities.StateProvinceID = sp.StateProvinceID
GROUP BY sp.StateProvinceName, month(o.OrderDate)
ORDER BY sp.StateProvinceName, month(o.OrderDate);
```

100 %

Results Messages

	StateProvinceName	month	AvgProcessDays
1	Alabama	1	10
2	Alabama	2	2
3	Alabama	3	3
4	Alabama	4	4
5	Alabama	5	3
6	Alabama	6	2
7	Alabama	7	11
8	Alabama	8	1
9	Alabama	9	3
10	Alabama	10	5
11	Alabama	11	11
12	Alabama	12	5
13	Alaska	1	1
14	Alaska	2	1
15	Alaska	3	1
16	Alaska	4	2
17	Alaska	5	5

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

```
WITH sold AS (
    SELECT
        si.StockItemID,
        SUM(ol.Quantity) AS QuantitySold
    FROM
        Warehouse.StockItems AS si
    LEFT JOIN Sales.OrderLines AS ol
    ON ol.StockItemID = si.StockItemID
    LEFT JOIN Sales.Orders AS o ON o.OrderID = ol.OrderID
    WHERE YEAR(o.OrderDate) = 2015
    GROUP BY si.StockItemID
),

purchases AS (
    SELECT
        si.StockItemName,
        si.StockItemID,
        SUM(pol.ReceivedOuters*si.QuantityPerOuter) as QuantityPurchased
    FROM Warehouse.StockItems AS si
```

100 %

Results Messages

	StockItemID	StockItemName	QuantityPurchased	QuantitySold
1	86	"The Gu" red shirt XML tag t-shirt (White) 5XL	7974804	21120
2	204	Tape dispenser (Red)	5665640	17220
3	98	"The Gu" red shirt XML tag t-shirt (Black) 4XL	8135496	22116
4	184	Shipping carton (Brown) 305x305x305mm	6217175	47725
5	77	"The Gu" red shirt XML tag t-shirt (White) XXS	7760580	19104
6	95	"The Gu" red shirt XML tag t-shirt (Black) XL	3913572	19992
7	78	"The Gu" red shirt XML tag t-shirt (White) XS	7996428	21972
8	80	"The Gu" red shirt XML tag t-shirt (White) M	3828444	21936
9	193	Black and orange glass with care despatch tape ...	11858376	40368

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 CTE AS (
    SELECT
        sub1.CustomerID
    FROM (
        SELECT
            o.CustomerID,
            o.OrderID,
            SUM(ol.PickedQuantity) as sold_num
        FROM Sales.Orders AS o
        INNER JOIN Sales.OrderLines AS ol
        ON o.OrderID = ol.OrderID AND YEAR(o.OrderDate) = 2016
        INNER JOIN Warehouse.StockItems AS s
        ON ol.StockItemID = s.StockItemID
        WHERE s.StockItemName LIKE '%mug%'
        GROUP BY o.CustomerID, o.OrderID
        ) AS sub1
    GROUP BY sub1.CustomerID
    HAVING SUM(sub1.sold_num) <= 10
)
```

100 %

Results Messages

	CustomerName	PhoneNumber	PrimaryContactName
1	Wingtip Toys (Mauldin, SC)	(803) 555-0100	Am Lo
2	Anand Mudaliyar	(206) 555-0100	Anand Mudaliyar
3	Bala Dixit	(209) 555-0100	Bala Dixit
4	Tailspin Toys (Ward Ridge, FL)	(239) 555-0100	Cristina Longo
5	Wingtip Toys (Plata, TX)	(210) 555-0100	Daniela Sal
6	Debbie Molina	(270) 555-0100	Debbie Molina
7	Wingtip Toys (Compass Lake, FL)	(239) 555-0100	Gireesh Bhogireddy
8	Wingtip Toys (Necedah, WI)	(262) 555-0100	Irene Sepp
9	Wingtip Toys (Salt Wells, NV)	(702) 555-0100	Jae-Hwa Min
10	Tailspin Toys (Manchester Center, VT)	(802) 555-0100	Karie Seymour

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

```
SELECT CityName FROM Application.Cities WHERE ValidFrom BETWEEN '2015-01-01' AND GETDATE();
```

100 %

Results Messages

	CityName
1	Adrian
2	Carlton
3	East Smithfield
4	Fairfax
5	Laupahoehoe
6	McWhorter
7	Norborne
8	North Granby
9	Pondosa
10	Richvale
11	Springville
12	Throop
13	Urbancrest

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
    si.StockItemName,
    c.DeliveryAddressLine1,
    c.DeliveryAddressLine2,
    sp.StateProvinceName,
    Cities.CityName,
    Countries.CountryName,
    c.CustomerName,
    c.PhoneNumber AS CustomerPhone,
    p.FullName AS ContactPersonName,
    ol.Quantity
FROM Sales.OrderLines AS ol
LEFT JOIN Sales.Orders AS o
ON ol.OrderID = o.OrderID
LEFT JOIN Sales.Customers AS c
ON c.CustomerID = o.CustomerID
LEFT JOIN Warehouse.StockItems AS si
ON si.StockItemID = ol.StockItemID
LEFT JOIN Application.Cities
ON Cities.CityID = c.DeliveryCityID
LEFT JOIN Application.StateProvinces AS sp
ON Cities.StateProvinceID = sp.StateProvinceID
LEFT JOIN Application.Countries
ON Countries.CountryID = sp.CountryID
LEFT JOIN Application.People AS p ON p.PersonID = c.PrimaryContactPersonID
WHERE o.OrderDate = '2014-07-01';
```

100 %

Results Messages

	StockItemName	DeliveryAddressLine1	DeliveryAddressLine2	StateProvinceName	CityName	CountryName	CustomerName	CustomerPhone	ContactPersonName	Qu
1	Developer joke mug - understanding recursion re...	Unit 212	1837 Vasiljevic Boulevard	Pennsylvania	Rose Tree	United States	Wingtip Toys (Rose Tree, PA)	(215) 555-0100	Lan Chu	7
2	Developer joke mug - that's a hardware problem ...	Unit 193	583 Aluri Road	Puerto Rico (US Territory)	Rosa Sánchez	United States	Wingtip Toys (Rosa Sánchez, PR)	(787) 555-0100	Selma Seppanen	7
3	Developer joke mug - there are 10 types of peop...	Suite 258	771 Kidambi Road	Texas	Lavon	United States	Tailspin Toys (Lavon, TX)	(210) 555-0100	Alba Ponce	2
4	DBA joke mug - two types of DBAs (White)	Suite 185	1492 Shah Road	Illinois	Stonefort	United States	Tailspin Toys (Stonefort, IL)	(217) 555-0100	Nazeena Hosseini	2
5	Developer joke mug - (hip, hip, array) (Black)	Shop 20	1046 Saucier Road	Ohio	East Fultonham	United States	Wingtip Toys (East Fultonham, OH)	(216) 555-0100	Masa Buecek	2
6	DBA joke mug - mind if I join you? (White)	Unit 50	519 Jogi Street	Puerto Rico (US Territory)	Indios	United States	Tailspin Toys (Indios, PR)	(787) 555-0100	Roxane Rastgu	2
7	DBA joke mug - you might be a DBA if (Black)	Shop 11	965 Horackova Lane	New York	Greig	United States	Kertu Sokk	(212) 555-0100	Kertu Sokk	2
8	DBA joke mug - daaaaaa-ta (White)	Suite 258	771 Kidambi Road	Texas	Lavon	United States	Tailspin Toys (Lavon, TX)	(210) 555-0100	Alba Ponce	9
9	Developer joke mug - when your hammer is C++ (W...	Unit 15	685 Pavel Road	Alabama	Southside	United States	Manca Hrastovsek	(205) 555-0100	Manca Hrastovsek	9
10	Developer joke mug - when your hammer is C++ (B...	Shop 104	1889 Smirnov Road	South Carolina	Sans Souci	United States	Tailspin Toys (Sans Souci, SC)	(803) 555-0100	Coralie Emond	9

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

```
WITH Purchased AS(
    SELECT
        sg.StockGroupName,
        SUM(pol.ReceivedOuters) AS QuantityPurchased
    FROM Purchasing.PurchaseOrderLines AS pol
    LEFT JOIN Warehouse.StockItemStockGroups AS sisg
    ON pol.StockItemID = sisg.StockItemID
    LEFT JOIN Warehouse.StockGroups AS SG
    ON sisg.StockGroupID = sg.StockGroupID
    GROUP BY sg.StockGroupName
),

Sold AS(
    SELECT
        sg.StockGroupName,
        SUM(ol.PickedQuantity) AS QuantityOrdered
    FROM Sales.OrderLines AS ol
    LEFT JOIN Warehouse.StockItemStockGroups AS sisg
    ON ol.StockItemID = sisg.StockItemID
    LEFT JOIN Warehouse.StockGroups AS sg
    ON sisg.StockGroupID = sg.StockGroupID
    GROUP BY sg.StockGroupName)

SELECT
    Purchased.StockGroupName,
```

100 %

Results Messages

	StockGroupName	QuantityPurchased	QuantityOrdered	QuantityRemaining
1	T-Shirts	7576727	1581492	5995235
2	USB Novelties	1580	81057	-79477
3	Packaging Materials	2702432	5696347	-2993915
4	Clothing	7577583	2624338	4953245
5	Novelty Items	4638	1168276	-1163638
6	Furry Footwear	587	395849	-395262
7	Mugs	1442	244010	-242568
8	Computing Novelties	7579764	1962719	5617045
9	Toys	1313	121360	-120047

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 ol.StockItemID, c.DeliveryCityID, COUNT(*) AS Delivery
    FROM Sales.OrderLines AS ol
        JOIN Sales.Orders AS o ON o.OrderID = ol.OrderID
        JOIN sales.Customers AS c ON o.CustomerID = c.CustomerID
    WHERE YEAR(o.OrderDate) = 2016
    GROUP BY ol.StockItemID, c.DeliveryCityID),

CTE2 AS(
    SELECT StockItemID, DeliveryCityID
    FROM (
        SELECT StockItemID, DeliveryCityID,
            DENSE_RANK() OVER(PARTITION BY DeliveryCityId ORDER BY Delivery DESC) AS rnk
        FROM CTE1) AS sub
    WHERE rnk = 1
)

SELECT
    Cities.CityName,
    ISNULL(s.StockItemName, 'No Sale') AS MostDeliveryItems
FROM CTE2 JOIN Warehouse.StockItems s
ON CTE2.StockItemID = s.StockItemID
RIGHT JOIN Application.Cities
ON CTE2.DeliveryCityID = Cities.CityID
```

100 %

Results Messages

	CityName	MostDeliveryItems
1	Abbottsburg	USB food flash drive - banana
2	Abbottsburg	DBA joke mug - you might be a DBA if (White)
3	Abbottsburg	Superhero action jacket (Blue) 5XL
4	Abbottsburg	32 mm Double sided bubble wrap 10m
5	Abbottsburg	10 mm Double sided bubble wrap 10m
6	Abbottsburg	Ride on toy sedan car (Pink) 1/12 scale
7	Abbottsburg	Ogre battery-powered slippers (Green) M
8	Absecon	Superhero action jacket (Blue) 3XS
9	Absecon	Ride on toy sedan car (Pink) 1/12 scale
10	Absecon	IT joke mug - hardware: part of the computer th...

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

```
SELECT
    OrderID
FROM Sales.Invoices
WHERE JSON_VALUE(ReturnedDeliveryData, '$.Events[1].Comment') IS NOT NULL
```

100 %

Results Messages

	OrderID
1	1807
2	1793
3	1803
4	1851
5	1871
6	1894
7	1907
8	1914
9	1883
10	1913

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

```
SELECT
    DISTINCT StockItemName
FROM Warehouse.StockItems
WHERE JSON_VALUE(CustomFields,'$.CountryOfManufacture') = 'China'
```

100 %

Results Messages

	StockItemName
1	USB missile launcher (Green)
2	USB rocket launcher (Gray)
3	Office cube periscope (Black)
4	DBA joke mug - mind if I join you? (White)
5	DBA joke mug - mind if I join you? (Black)
6	DBA joke mug - daaaaaa-ta (White)
7	DBA joke mug - daaaaaa-ta (Black)
8	DBA joke mug - you might be a DBA if (White)
9	DBA joke mug - you might be a DBA if (Black)
10	DBA joke mug - it depends (White)

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

```
SELECT
    JSON_VALUE(CustomFields, '$.CountryOfManufacture') AS CountryOfManufacture,
    SUM(ol.Quantity) AS TotalQuantityStockSold
FROM Sales.Orders AS o
LEFT join Sales.OrderLines AS ol
ON o.OrderID = ol.OrderID
LEFT JOIN Warehouse.StockItems AS s
ON ol.StockItemID = s.StockItemID
WHERE YEAR(o.OrderDate) = 2015
GROUP BY JSON_VALUE(CustomFields, '$.CountryOfManufacture');
```

100 %

Results Messages

	CountryOfManufacture	TotalQuantityStockSold
1	Japan	22365
2	China	2850885

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 StockQuanlityByYear AS
    SELECT StockGroupName,[2013],[2014],[2015],[2016],[2017] FROM(
        SELECT
            sg.StockGroupName,
            YEAR(o.OrderDate) AS years,
            SUM(ol.Quantity) AS total_quantity
        FROM Sales.OrderLines AS ol
        LEFT JOIN sales.Orders AS o
        ON ol.OrderID = o.OrderID
        LEFT JOIN Warehouse.StockItems AS si
        ON ol.StockItemID = si.StockItemID
        LEFT JOIN Warehouse.StockItemStockGroups AS ssg
        ON ssg.StockItemID = ol.StockItemID
        LEFT JOIN Warehouse.StockGroups AS sg
        ON sg.StockGroupID = ssg.StockGroupID
        WHERE YEAR(o.OrderDate) IN (2013, 2014, 2015, 2016, 2017)
        GROUP BY sg.StockGroupName, YEAR(o.OrderDate)
    ) AS p
    PIVOT (
    SUM(p.total_quantity)
    FOR years IN ([2013],[2014],[2015],[2016],[2017])
    ) AS pivotTable;
```

110 %

Results Messages

	StockGroupName	2013	2014	2015	2016	2017
1	T-Shirts	486924	528096	558144	226908	NULL
2	USB Novelties	21328	23685	26048	9996	NULL
3	Packaging Materials	1572415	1694778	1826433	744417	NULL
4	Clothing	767341	831573	889178	354826	NULL
5	Novelty Items	276609	306077	328677	256913	NULL
6	Furry Footwear	107839	112845	125924	49241	NULL
7	Mugs	65713	70384	77268	30645	NULL
8	Computing Novelties	588555	639315	677480	275949	NULL
9	Toys	32266	35403	38303	15388	NULL

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]

```
CREATE VIEW StockQuanlityByYear2 AS
    SELECT
        Years, [Clothing], [USB Novelties], [Computing Novelties], [Novelty Items],[T-Shirts], [Mugs],[Furry Footwear],[Toys], [Packaging Materials]
    FROM (
        SELECT YEAR(o.OrderDate) AS Years, sg.StockGroupName AS StockGroupName, sum(ol.Quantity) AS TotalQuantity
        FROM Sales.OrderLines AS ol
        LEFT JOIN sales.Orders AS o
        ON ol.OrderID = o.OrderID
        LEFT JOIN Warehouse.StockItems AS si
        ON ol.StockItemID = si.StockItemID
        LEFT JOIN Warehouse.StockItemStockGroups AS ssg
        ON ssg.StockItemID = ol.StockItemID
        LEFT JOIN Warehouse.StockGroups AS sg
        ON sg.StockGroupID = ssg.StockGroupID
        WHERE YEAR(o.OrderDate) IN ( 2013, 2014, 2015, 2016, 2017)
        GROUP BY sg.StockGroupName, YEAR(o.OrderDate)
    ) AS p
    PIVOT (
    SUM(p.TotalQuantity)
    FOR StockGroupName IN ([Clothing], [USB Novelties], [Computing Novelties], [Novelty Items],[T-Shirts], [Mugs],[Furry Footwear],[Toys], [Packaging Materials])
    ) AS pivotTable
    ORDER BY Years
```

sults Messages

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

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 Sales.OrderTotal (@orderid INT)
RETURNS TABLE AS
RETURN (
    SELECT OrderID, SUM(Quantity * UnitPrice) AS Total
    FROM Sales.OrderLines
    WHERE OrderID = @orderid
    GROUP BY OrderID
)

SELECT * FROM Sales.Invoices CROSS APPLY Sales.OrderTotal(OrderID)
```

ults Messages

InvoiceID	CustomerID	BillToCustomerID	OrderID	DeliveryMethodID	ContactPersonID	AccountsPersonID	SalespersonPersonID	PackedByPersonID	InvoiceDate	CustomerPurchaseOrderNumber
1	832	832	1	3	3032	3032	2	14	2013-01-01	12126
2	803	803	2	3	3003	3003	8	14	2013-01-01	15342
3	105	1	3	3	1209	1001	7	14	2013-01-01	12211
4	57	1	4	3	1113	1001	16	14	2013-01-01	17129
5	905	905	5	3	3105	3105	3	14	2013-01-01	10369
6	976	976	6	3	3176	3176	13	14	2013-01-01	13383
7	575	401	7	3	2349	2001	8	14	2013-01-01	17913
8	964	964	8	3	3164	3164	7	14	2013-01-01	14518
9	77	1	9	3	1153	1001	7	14	2013-01-01	17577
10	191	1	10	3	1381	1001	20	14	2013-01-01	18030

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 PRIMARY KEY,
OrderDate DATE,
OrderTotal DECIMAL(18, 2),
CustomerID INT)

GO

CREATE PROCEDURE ods.OrderTotalOfDate
    @OrderDate DATE
AS

IF EXISTS (SELECT 1 FROM ods.Orders WHERE OrderDate = @OrderDate)
    BEGIN
        RAISERROR('Date Exists ', 16, 1)
    END
ELSE
    BEGIN
        BEGIN TRANSACTION
```

```
            INSERT INTO ods.Orders
            SELECT o.OrderID, o.OrderDate, f.Total, o.CustomerID
            FROM Sales.Orders o
                CROSS APPLY Sales.OrderTotal(OrderID) f
            WHERE o.OrderDate = @OrderDate
        COMMIT
    END
GO
EXEC ods.OrderTotalOfDate '2013-01-01'
EXEC ods.OrderTotalOfDate '2013-01-02'
EXEC ods.OrderTotalOfDate '2013-01-03'
EXEC ods.OrderTotalOfDate '2013-01-04'
EXEC ods.OrderTotalOfDate '2013-01-05'
```

Messages

```
(79 rows affected)

(74 rows affected)

(83 rows affected)

(42 rows affected)

(50 rows affected)

Completion time: 2022-11-04T06:00:44.7453636-07:00
```

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(20),
    [Range] NVARCHAR(20),
    [Shelflife] NVARCHAR(20)
)

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(CustomFields,'$.CountryOfManufacture') AS CountryOfManufacture,
    JSON_VALUE(CustomFields,'$.Range') AS [Range],
    JSON_VALUE(CustomFields,'$.ShelfLife') AS Shelflife
   FROM WideWorldImporters.Warehouse.StockItems

  (227 rows affected)
```

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 PROCEDURE ods.NewOrderTotalOfDate
    @OrderDate DATE
AS

BEGIN TRANSACTION
    DELETE FROM ods.Orders
    WHERE OrderDate < @OrderDate
COMMIT

BEGIN TRANSACTION
    MERGE ods.Orders T
    USING (
            SELECT o.OrderID, o.OrderDate, f.Total, o.CustomerID
            FROM Sales.Orders o
                CROSS APPLY Sales.OrderTotal(OrderID) f
            WHERE DATEDIFF(d, @OrderDate, OrderDate) BETWEEN 1 AND 7
            ) R
    ON T.OrderID = R.OrderID
    WHEN NOT MATCHED
        THEN INSERT VALUES (R.OrderID, R.OrderDate, R.Total, R.CustomerID);
      COMMIT
```

```
Commands completed successfully.
```

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

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

Transaction functions as SQL's save/undo button. It's a logical work unit that performs one or more activities. It needs to conform the ACID principle: Atomicity, consistency, isolation, and durability. There are two outcomes: committed (which means permanently saves the changes made by the current transaction) and rolled back (which mean undo the changes made by the current transaction). There are different types of transactions. First, Autocommit transaction is the default type for SQL Sever: each T-SQL statement is evaluated as a transaction and committed or rolled back according to their results. The successful/failed statements are committed/rolled back accordingly. Second, Implicit Transaction enables SQL Server to start an implicit transaction for every DML statement, but explicit commit or rollback commands are needed at the end of the statements. You don't need to specify BEGIN and END. May cause performance hits. Third, Explicit Transaction define a transaction exactly with the starting and ending points of the transaction. You need to specify where a transaction BEGIN and END.

Isolation Levels: Isolation levels define the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system. A transaction isolation level is defined by the following concurrency issues: Dirty Read, Lost Updates, Non-Repeatable Reads, and Phantom Reads. Based on these phenomena, SQL standards defines five isolation levels for concurrency controls:

1. Read Uncommitted:
   - Lowest level of isolation, pessimistic
   - One transaction is allowed to read the data that is about to be changed by the commit of another process. Allow the dirty read problem because has not committed yet
2. Read Committed
   - second level of isolation, pessimistic
   - only allowed to read data that is committed, so it can eliminate the dirty read problem
3. Repeatable Read
   - second level of isolation, pessimistic
   - Transaction has to wait till another transaction's update or read query is complete. But if there's an insert transaction, it does not wait for anyone. This eliminates the Non-Repeatable Read Problem but can lead to Phantom Read.
4. Serializable
   - highest level of isolation, pessimistic
   - We can ask any transaction to wait until the current transaction completes, so by implementing this level of isolation, we can prevent Phantom Read Problem
5. Snapshot
   - Only one considered as optimistic
   - Avoids most locking and blocking by using row versioning
   - When data is modified, the committed versions of affected rows are copied to tempdb and given version number. This operation is called copy on write and used for all inserts, updates, and deletes using this technique
   - When another session reads the same data, the committed version of the data as of the time the reading transaction began is returned

Locks: Transactions use locks to control concurrent access to data, achieving two important database goals: Consistency ensures that the data you are viewing or changing is not changed by other users until you are finished with the data. Locks has different types:

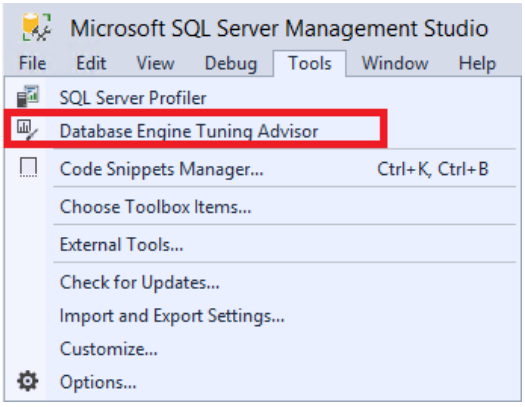
1. Exclusive(X) and Shared(S)
   - will ensure that a page or row will be reserved exclusively for the transaction that imposed the exclusive lock, as long as the transaction holds the lock
2. Shared (S)
   - will reserve a page or row to be available only for reading
   - can be imposed by several transactions at the same time.
   - will allow write operations, but no DDL changes will be allowed
3. Update (U)
   - similar to an exclusive lock but is designed to be more flexible in a way.
   - can be imposed on a record that already has a shared lock. In such a case, the update lock will impose another shared lock on the target row. Once the transaction that holds the update lock is ready to change the data, the update lock (U) will be transformed to an exclusive lock (X).
4. Intent (I)
   - used by a transaction to inform another transaction about its intention to acquire a lock
   - The purpose of such lock is to ensure data modification to be executed properly by preventing another transaction to acquire a lock on the next in hierarchy object.
5. Schema (Sch)
6. Bulk update (BU)

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.

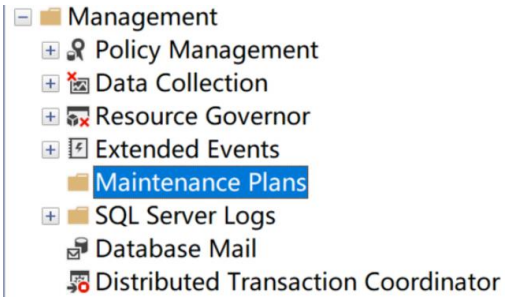
Tuning Advisor

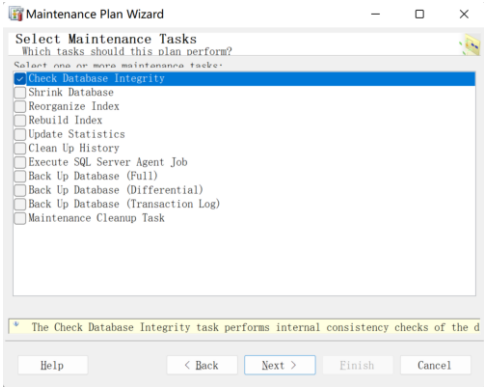
Reference: https://learn.microsoft.com/en-us/sql/tools/dta/lesson-2-using-database-engine-tuning-advisor?view=sql-server-ver16

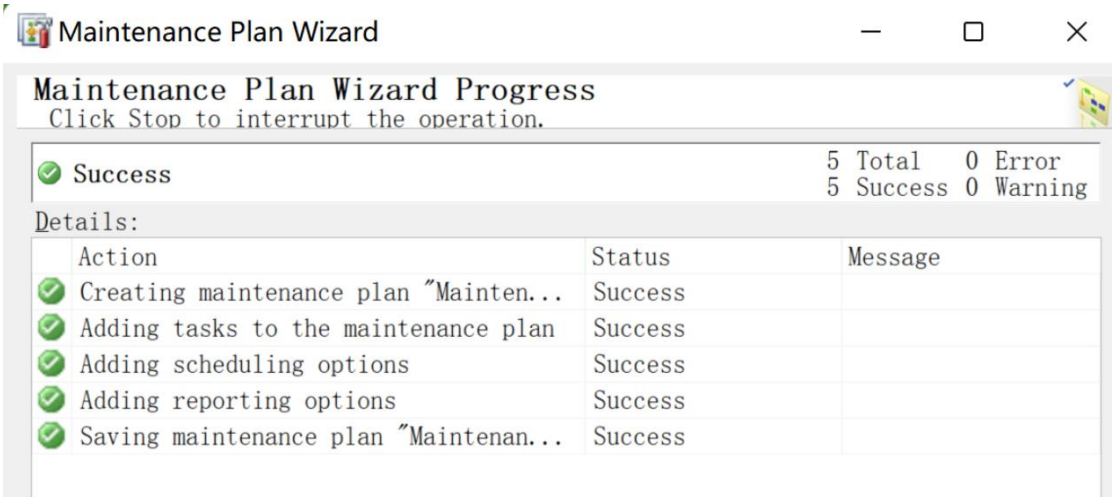
Tuning Advisor examines how queries are processed in the databases you specify, and then recommends how you can improve query processing performance by modifying database structures such as indexes, indexed views, and partitioning.



Execution Plan





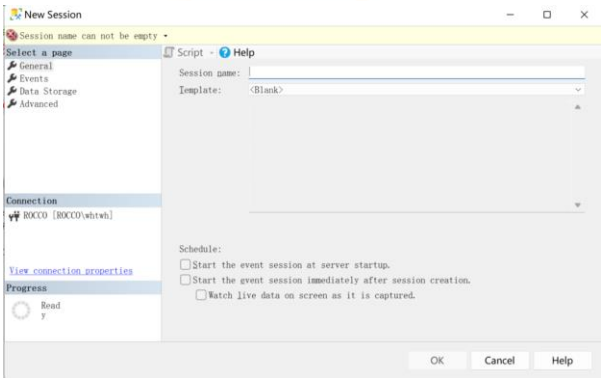


Extended Events:

Extended Events provides the ability to monitor and collect different events and system information from SQL Server

Reference:

https://www.sqlshack.com/using-sql-server-extended-events-to-monitor-query-performance/



DMV

Dynamic management views and functions return server state information that can be used to monitor the health of a server instance, diagnose problems, and tune performance.

```
SELECT login_name ,COUNT(session_id) AS session_count
FROM sys.dm_exec_sessions
GROUP BY login_name;
```

sults Messages

login_name	session_count
NT SERVICE\SQLTELEMETRY	1
ROCCO\whtwh	5
sa	121

This returns current session information such as Session ids, login information like name, host, client info. It also returns some performance information like memory usage, CPU time, reads, writes and lots of other useful info.

```
SELECT  c.session_id, c.net_transport, c.encrypt_option, c.auth_scheme, s.host_name, s.program_name,
s.client_interface_name, s.login_name, s.nt_domain, s.nt_user_name, s.original_login_name, c.connect_time,
s.login_time
FROM sys.dm_exec_connections AS c
JOIN sys.dm_exec_sessions AS s
ON c.session_id = s.session_id
WHERE c.session_id = @@SPID -- @@SPID returns your current session SPID
```

sults Messages

session_id	net_transport	encrypt_option	auth_scheme	host_name	program_name	client_interface_name	login_name	nt_domain	nt_user_name	original_login_name	connect_time
124	Shared memory	FALSE	NTLM	ROCCO	Microsoft SQL Server Management Studio - Query	.Net SqlClient Data Provider	ROCCO\whtwh	ROCCO	whtwh	MicrosoftAccount\whtwht97@hotmail.com	2022-11-03 1

This DMV gives information like when the connection started, last read, last write, client net address most recent sql_handle and other network connection related information

Logs

- SQL Server Logs
  - Current - 11/27/2021 6:25:00 PM
  - Archive #1 - 11/27/2021 1:36:00 AM
  - Archive #2 - 11/26/2021 12:52:00 AM
  - Archive #3 - 11/24/2021 11:24:00 PM
  - Archive #4 - 11/24/2021 2:49:00 AM
  - Archive #5 - 11/23/2021 12:07:00 AM
  - Archive #6 - 11/22/2021 12:26:00 AM

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.

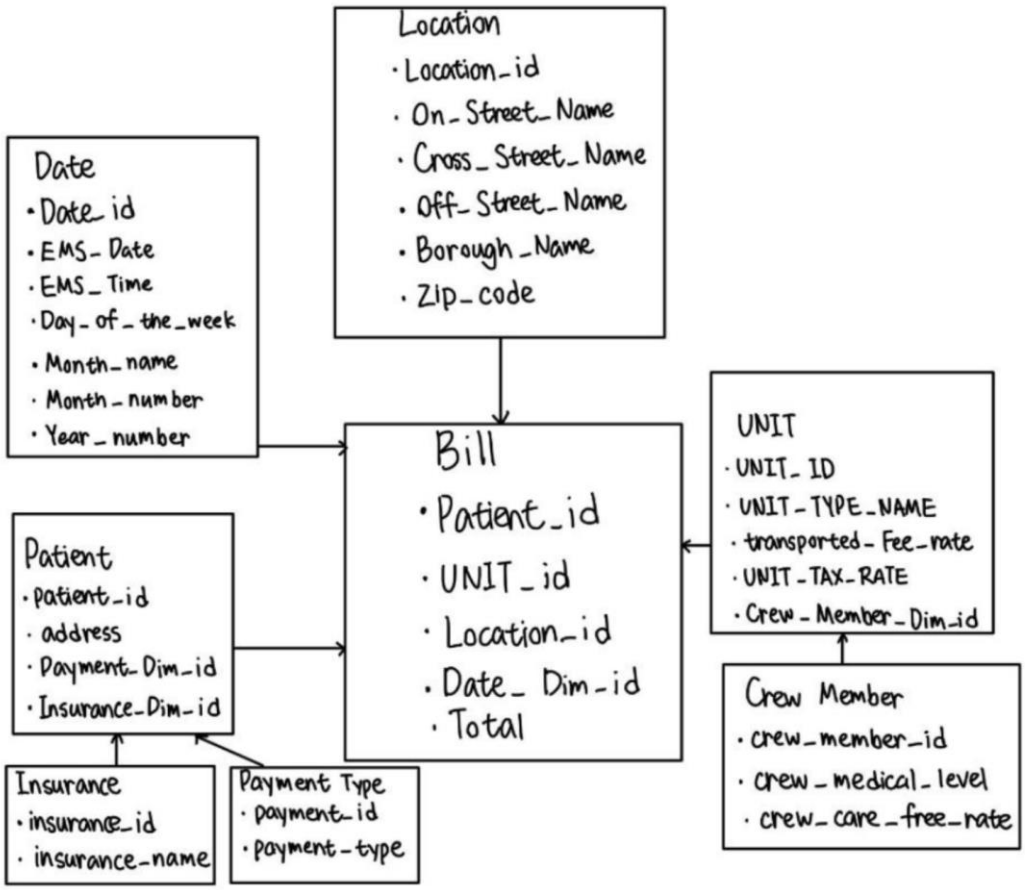
```
-- In the begining of the merge, we have to insert our bike products' info into the
Warehouse.StockItems table. And then, we can assign these bike
-- products into their group in StockGroups table.
INSERT INTO Application.People (
 FullName, PreferredName, SearchName, IsPermittedToLogon,
 LogonName, PhoneNumber, EmailAddress, FaxNumber, LastEditedBy,
 ValidFrom, ValidTo)
SELECT
 aau.FullName, aau.PreferredName, aau.SearchName, aau.IsPermittedToLogon,
 aau.LogonName, aau.PhoneNumber, aau.EmailAddress, aau.FaxNumber,


  aau.LastEditedBy, aau.ValidFrom, aau.ValidTo
 FROM [Adventure_works].Application.Users aau

 --Insert basic new product info into WWI database
 INSERT INTO Warehouse.StockItems (
  StockItemID, ColorID, StockItemName, Size, TaxRate, UnitPrice, Tags)
 SELECT
  aap.Product_id, aap.ColorID, aap.product_name,
  aap.size, aap.TaxRate, aap.UnitPrice, aap.Category
 FROM [Adventure_works].Application.Products aap

 -- Assign product group into WWI stock_group
 -- Firstly, a new StockGroupID, Stockname, LastEditedBy, ValidFrom, ValidTo info need in
the [WWI].Warehouse.StockGroups needed
 INSERT INTO Warehouse.StockGroups(StockGroupID, StockGroupName, LastEditedBy,
ValidFrom, ValidTo)
 VALUE (11, 'Bike', 1, '2022-08-03 00:00:00.0000000', '9999-12-31 23:59:59.9999999')

 --Then we can assign different bike products into their groups in the
StockItemStockGroup table
 INSERT INTO Warehouse.StockItemStockGroups
 SELECT DISTINCT aap.Product_id, 11, 1, '2022-08-03 00:00:00.0000000'
 FROM [Adventure_works].Application.Products aap
```


Location
· Location_id
· On_Street_Name
· Cross_Street_Name
· Off_Street_Name
· Borough_Name
· Zip_code
Date
· Date_id
· EMS_Date
· EMS_Time
· Day_of_the_week
· Month_name
· Month_number
· Year_number
Bill
· Patient_id
· UNIT_id
· Location_id
· Date_Dim_id
· Total
UNIT
· UNIT_ID
· UNIT_TYPE_NAME
· transported_Fee_rate
· UNIT_TAX_RATE
· Crew_Member_Dim_id
Patient
· Patient_id
· address
· Payment_Dim_id
· Insurance_Dim_id
Crew Member
· crew_member_id
· crew_medical_level
· crew_care_free_rate
Insurance
· insurance_id
· insurance_name
Payment Type
· payment_id
· payment_type

```sql
ALTER TABLE WideWorldImportersDW.Dimension.[Stock Item]
ADD [Country of Manufacture] NVARCHAR(20)

UPDATE WideWorldImportersDW.Dimension.[Stock Item]
SET [Country of Manufacture] =
JSON_VALUE(SI.CustomFields,'$.CountryOfManufacture')
FROM WideWorldImporters.Warehouse.StockItems AS SI
WHERE [Stock Item Key] = SI.StockItemID

SELECT [Country of Manufacture] FROM WideWorldImportersDW.Dimension.[Stock Item]

CREATE PROCEDURE dbo.ExtractOrder
AS
	SELECT
	C.DeliveryCityID,
	O.CustomerID,
	OL.StockItemID ,
	O.OrderDate,
	CONVERT(DATE,O.PickingCompletedWhen) AS [Picked Date Key],
	O.SalespersonPersonID,
	O.PickedByPersonID,
	O.OrderID ,
	O.BackorderOrderID ,
	SI.StockItemName,
	PT.PackageTypeName,
	OL.Quantity,
	OL.UnitPrice,
	OL.TaxRate,
	IL.TaxAmount
	FROM WideWorldImporters.Sales.Orders AS O
	JOIN WideWorldImporters.Sales.OrderLines AS OL
	ON O.OrderID = OL.OrderID
	JOIN WideWorldImporters.Sales.Invoices AS I
	ON I.OrderID = O.OrderID
	JOIN WideWorldImporters.Sales.InvoiceLines AS IL
	ON IL.InvoiceID = I.InvoiceID AND IL.StockItemID = OL.StockItemID
	JOIN WideWorldImporters.Warehouse.StockItems AS SI
	ON SI.StockItemID = OL.StockItemID
	JOIN WideWorldImporters.Warehouse.PackageTypes AS PT
	ON PT.PackageTypeID = OL.PackageTypeID
	JOIN WideWorldImporters.Sales.Customers AS C
	ON C.CustomerID = O.CustomerID

GO

CREATE TABLE WideWorldImportersDW.Integration.ExtractOrder_Staging(
	DeliveryCityID INT,
	CustomerID INT,
	StockItemID INT ,
	OrderDate DATE ,
	[Picked Date Key] DATE ,
	SalespersonPersonID INT ,
	PickedByPersonID INT ,
	OrderID INT,
	BackorderOrderID INT ,
	StockItemName NVARCHAR(MAX),
	PackageTypeName NVARCHAR(50),
	Quantity INT,
```

```
       UnitPrice DECIMAL(18,2),
       TaxRate DECIMAL(18,3),
       TaxAmount DECIMAL(18,2)
);

INSERT INTO WideWorldImportersDW.Integration.ExtractOrder_Staging
    EXEC dbo.ExtractOrder ;

CREATE PROCEDURE dbo.TrasformOrder
AS
        SELECT
        DeliveryCityID,
        CustomerID,
        StockItemID,
        OrderDate,
        [Picked Date Key],
        SalespersonPersonID,
        PickedByPersonID,
        OrderID,
        BackorderOrderID,
        StockItemName,
        PackageTypeName,
        Quantity,
        UnitPrice,
        TaxRate,
        Quantity*UnitPrice  AS [Total Excluding Tax],
        TaxAmount,
        Quantity*UnitPrice + TaxAmount AS [Total Including Tax]
        FROM WideWorldImportersDW.Integration.ExtractOrder_Staging

GO

CREATE TABLE WideWorldImportersDW.Integration.TransformOrder_Staging(
       DeliveryCityID INT,
       CustomerID INT,
       StockItemID INT ,
       OrderDate DATE ,
       [Picked Date Key] DATE ,
       SalespersonPersonID INT ,
       PickedByPersonID INT ,
       OrderID INT,
       BackorderOrderID INT ,
       StockItemName NVARCHAR(MAX),
       PackageTypeName NVARCHAR(50),
       Quantity INT,
       UnitPrice DECIMAL(18,2),
       TaxRate DECIMAL(18,3),
       [Total Excluding Tax] DECIMAL(18,3),
       TaxAmount DECIMAL(18,2),
       [Total Including Tax] DECIMAL(18,3)
);

INSERT INTO  WideWorldImportersDW.Integration.TransformOrder_Staging
       EXEC dbo.TrasformOrder;

DROP TABLE WideWorldImportersDW.Integration.ExtractOrder_Staging;

CREATE PROCEDURE dbo.LoadOrder
```

```
AS
       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.[City Key],
        ISNULL(C.[Customer Key],0) AS [Customer Key],
        SI.[Stock Item Key],
        OrderDate AS [Order Date Key],
        [Picked Date Key],
        E.[Employee Key] AS [Salesperson Key],
        EE.[Employee Key] AS [Picker Key],
        OrderID AS [WWI Order ID],
        BackorderOrderID AS [WWI Backorder ID],
        StockItemName AS [Description],
        PackageTypeName AS [Package],
        Quantity,
        UnitPrice AS [Unit Price],
        TaxRate AS [Tax Rate],
        [Total Excluding Tax],
        TaxAmount AS [Tax Amount],
        [Total Including Tax],
        9
       FROM WideWorldImportersDW.Integration.TransformOrder_Staging    AS A
       LEFT JOIN WideWorldImportersDW.Dimension.Customer AS C
       ON A.CustomerID = C.[WWI Customer ID] AND C.[Valid To]='9999-12-31
23:59:59.9999999'
       LEFT JOIN WideWorldImportersDW.Dimension.[Stock Item] AS SI
       ON SI.[WWI Stock Item ID] = A.StockItemID AND SI.[Valid To] = '9999-12-31
23:59:59.9999999'
       LEFT JOIN WideWorldImportersDW.Dimension.Employee AS E
       ON E.[WWI Employee ID] = A.SalespersonPersonID AND E.[Valid To] = '9999-12-
31 23:59:59.9999999'
       LEFT JOIN WideWorldImportersDW.Dimension.Employee AS EE
       ON EE.[WWI Employee ID] = A.PickedByPersonID AND EE.[Valid To] = '9999-12-
31 23:59:59.9999999'
       LEFT JOIN WideWorldImportersDW.Dimension.City AS City
       ON City.[WWI City ID] = A.DeliveryCityID AND City.[Valid To] = '9999-12-31
23:59:59.9999999'
```

```
EXEC dbo.LoadOrder
DROP TABLE WideWorldImportersDW.Integration.TransformOrder_Staging;
```