SQL Assignment

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).

```
USE WideWorldImporters
GO

--1
SELECT p.FullName,p.PhoneNumber,p.FaxNumber, c.CustomerName as Company_Name
FROM Application.People p JOIN Sales.Customers c ON p.PersonID = c.CustomerID
```

100 %

Results | Messages

	FullName	PhoneNumber	FaxNumber	Company_Name
1	Data Conversion Only	NULL	NULL	Tailspin Toys (Head Office)
2	Kayla Woodcock	(415) 555-0102	(415) 555-0103	Tailspin Toys (Sylvanite, MT)
3	Hudson Onslow	(415) 555-0102	(415) 555-0103	Tailspin Toys (Peeples Valley, AZ)
4	Isabella Rupp	(415) 555-0102	(415) 555-0103	Tailspin Toys (Medicine Lodge, KS)
5	Eva Muirden	(415) 555-0102	(415) 555-0103	Tailspin Toys (Gasport, NY)
6	Sophia Hinton	(415) 555-0102	(415) 555-0103	Tailspin Toys (Jessie, ND)
7	Amy Trefl	(415) 555-0102	(415) 555-0103	Tailspin Toys (Frankewing, TN)
8	Anthony Grosse	(415) 555-0102	(415) 555-0103	Tailspin Toys (Bow Mar, CO)
9	Alica Fatnowna	(415) 555-0102	(415) 555-0103	Tailspin Toys (Netcong, NJ)
10	Stella Rosenhain	(415) 555-0102	(415) 555-0103	Tailspin Toys (Wimbledon, ND)
11	Ethan Onslow	(415) 555-0102	(415) 555-0103	Tailspin Toys (Devault, PA)
12	Henry Forlonge	(415) 555-0102	(415) 555-0103	Tailspin Toys (Biscay, MN)
13	Hudson Hollinworth	(415) 555-0102	(415) 555-0103	Tailspin Toys (Stonefort, IL)
14	Lily Code	(415) 555-0102	(415) 555-0103	Tailspin Toys (Long Meadow, MD)
15	Taj Shand	(415) 555-0102	(415) 555-0103	Tailspin Toys (Batson, TX)
16	Archer Lamble	(415) 555-0102	(415) 555-0103	Tailspin Toys (Coney Island, MO)

Query executed su... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 107 rows

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

```
--2
SELECT c.CustomerName AS Company_Name
FROM Sales.Customers c LEFT JOIN Application.People p ON c.PrimaryContactPersonID = p.PersonID
WHERE c.PhoneNumber = p.PhoneNumber;
```

100 %

Results | Messages

	Company_Name
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)

Query executed succ... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 663 rows

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

```sql
--3
SELECT CustomerName
FROM(SELECT c.CustomerID, o.OrderDate,c.CustomerName,
     ROW_NUMBER() OVER(PARTITION BY c.CustomerID ORDER BY o.OrderDate desc) as rank_num
     FROM Sales.Customers c JOIN Sales.Orders o ON c.CustomerID = o.CustomerID) new_table
WHERE rank_num = 1 and OrderDate < '2016-01-01'
```

100 %

Results Messages

	CustomerName

Query executed succ... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 0 rows

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

```sql
--4
SELECT StockItemID, sum(ReceivedOuters*QuantityPerOuter) AS Total_Quantity
FROM (SELECT s.StockItemID, p1.ReceivedOuters,s.QuantityPerOuter
      FROM Purchasing.PurchaseOrderLines p1 JOIN Warehouse.StockItems s ON p1.StockItemID = s.StockItemID
      JOIN Purchasing.PurchaseOrders p2 ON p1.PurchaseOrderID = p2.PurchaseOrderID
      WHERE year(p2.OrderDate) = 2013) AS A
GROUP BY StockItemID;
```

100 %

Results Messages

	StockItemID	Total_Quantity
1	23	30
2	46	44
3	215	18
4	69	50
5	192	396
6	92	72

Query executed su... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 219 rows

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

```sql
--5
SELECT DISTINCT StockItemID
FROM Purchasing.PurchaseOrderLines
WHERE len(Description) >= 10;
```

100 %

Results Messages

	StockItemID
1	23
2	46
3	215
4	69
5	192

Query executed succ... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 227 rows

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

```
--6
SELECT DISTINCT s1.StockItemID,s1.StockItemName
FROM Warehouse.StockItems s1 JOIN Warehouse.StockItemTransactions s2 ON s1.StockItemID = s2.StockItemID
JOIN Purchasing.Suppliers p ON s1.SupplierID = p.SupplierID
JOIN Application.Cities c ON p.DeliveryCityID = c.CityID
JOIN Application.StateProvinces sp ON c.StateProvinceID = sp.StateProvinceID
WHERE year(s2.TransactionOccurredWhen) = 2014 and sp.StateProvinceName not in ('Alabama' , 'Georgia');
```

100 %

Results | Messages

	StockItemID	StockItemName
1	23	DBA joke mug - it depends (Black)
2	215	Air cushion machine (Blue)
3	46	Developer joke mug - a foo walks into a bar (Wh...
4	192	Black and orange fragile despatch tape 48mmx100m
5	69	Ride on toy sedan car (Blue) 1/12 scale

Query executed succ... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:04 | 219 rows

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

```
--7
WITH Delivery as (SELECT sp.StateProvinceName, DATEDIFF(DAY,o.OrderDate,
                  cast(i.ConfirmedDeliveryTime as date)) as delivery_day
                  FROM Sales.Orders o JOIN Sales.Invoices i ON i.OrderID = o.OrderID
                  JOIN Sales.Customers c ON o.CustomerID = c.CustomerID
                  JOIN Application.Cities ct ON c.DeliveryCityID = ct.CityID
                  FULL JOIN Application.StateProvinces sp ON ct.StateProvinceID = sp.StateProvinceID)

SELECT StateProvinceName, avg(delivery_day) as Average_delivery_day
FROM Delivery
GROUP BY StateProvinceName
ORDER BY Average_delivery_day DESC;
```

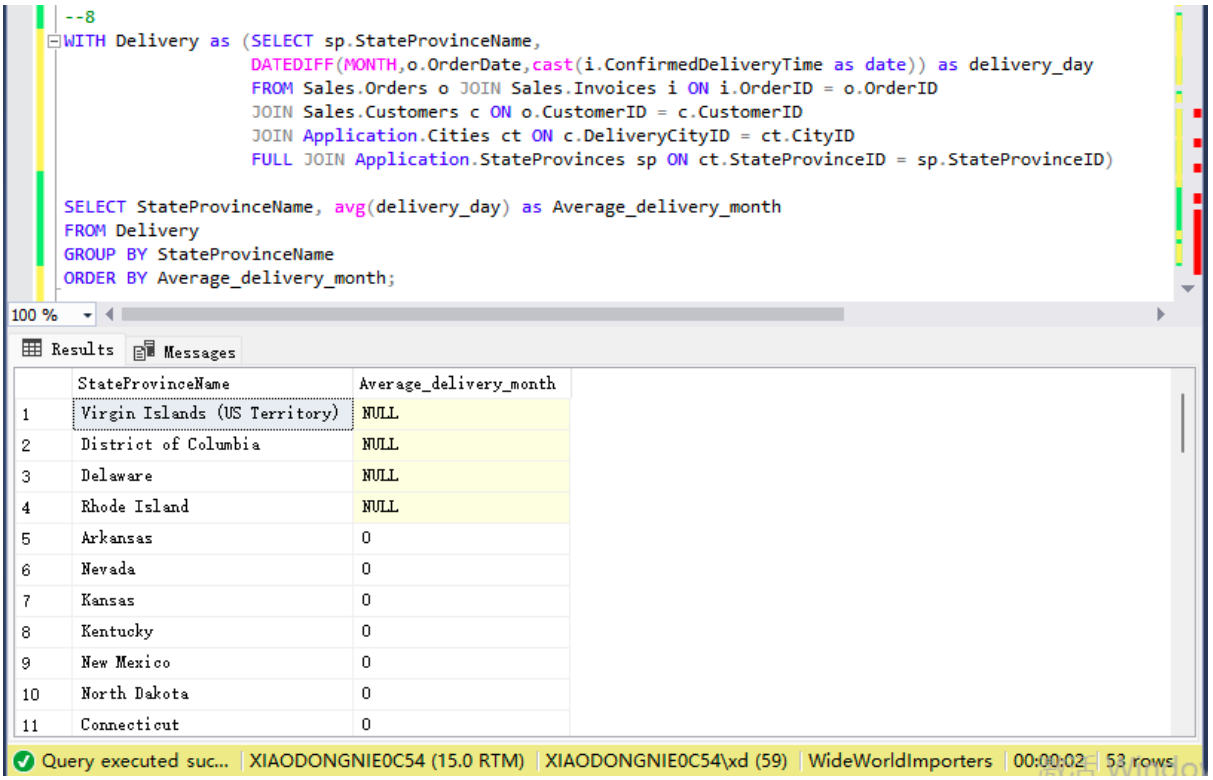
100 %

Results | Messages

	StateProvinceName	Average_delivery_day
1	New Hampshire	10
2	Wyoming	8
3	Indiana	7
4	Michigan	6
5	North Carolina	6
6	Nebraska	6
7	Arizona	6
8	Virginia	6
9	Missouri	6
10	South Carolina	6
11	West Virginia	6

Query executed suc... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:02 | 53 rows

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



```
--8
WITH Delivery as (SELECT sp.StateProvinceName,
                  DATEDIFF(MONTH,o.OrderDate,cast(i.ConfirmedDeliveryTime as date)) as delivery_day
                  FROM Sales.Orders o JOIN Sales.Invoices i ON i.OrderID = o.OrderID
                  JOIN Sales.Customers c ON o.CustomerID = c.CustomerID
                  JOIN Application.Cities ct ON c.DeliveryCityID = ct.CityID
                  FULL JOIN Application.StateProvinces sp ON ct.StateProvinceID = sp.StateProvinceID)

SELECT StateProvinceName, avg(delivery_day) as Average_delivery_month
FROM Delivery
GROUP BY StateProvinceName
ORDER BY Average_delivery_month;
```

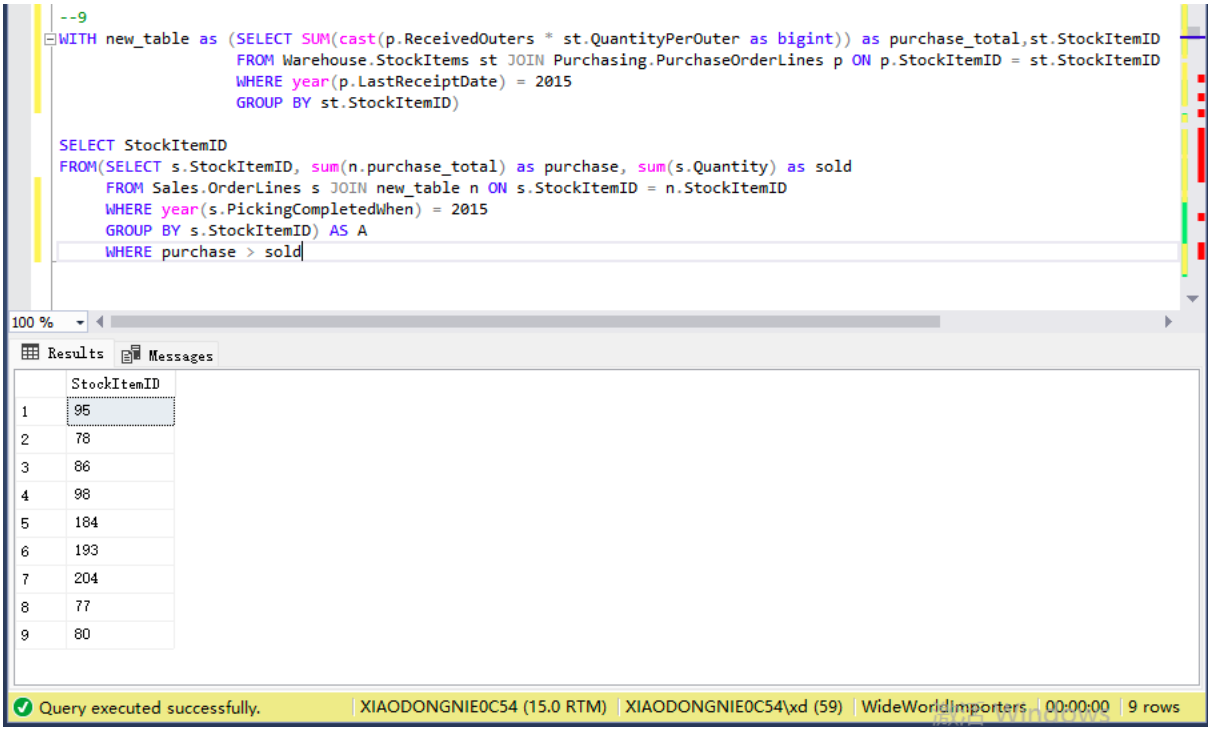
100 %

Results | Messages

	StateProvinceName	Average_delivery_month
1	Virgin Islands (US Territory)	NULL
2	District of Columbia	NULL
3	Delaware	NULL
4	Rhode Island	NULL
5	Arkansas	0
6	Nevada	0
7	Kansas	0
8	Kentucky	0
9	New Mexico	0
10	North Dakota	0
11	Connecticut	0

Query executed suc... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:02 | 53 rows

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



```
--9
WITH new_table as (SELECT SUM(cast(p.ReceivedOuters * st.QuantityPerOuter as bigint)) as purchase_total,st.StockItemID
                   FROM Warehouse.StockItems st JOIN Purchasing.PurchaseOrderLines p ON p.StockItemID = st.StockItemID
                   WHERE year(p.LastReceiptDate) = 2015
                   GROUP BY st.StockItemID)

SELECT StockItemID
FROM(SELECT s.StockItemID, sum(n.purchase_total) as purchase, sum(s.Quantity) as sold
     FROM Sales.OrderLines s JOIN new_table n ON s.StockItemID = n.StockItemID
     WHERE year(s.PickingCompletedWhen) = 2015
     GROUP BY s.StockItemID) AS A
     WHERE purchase > sold
```

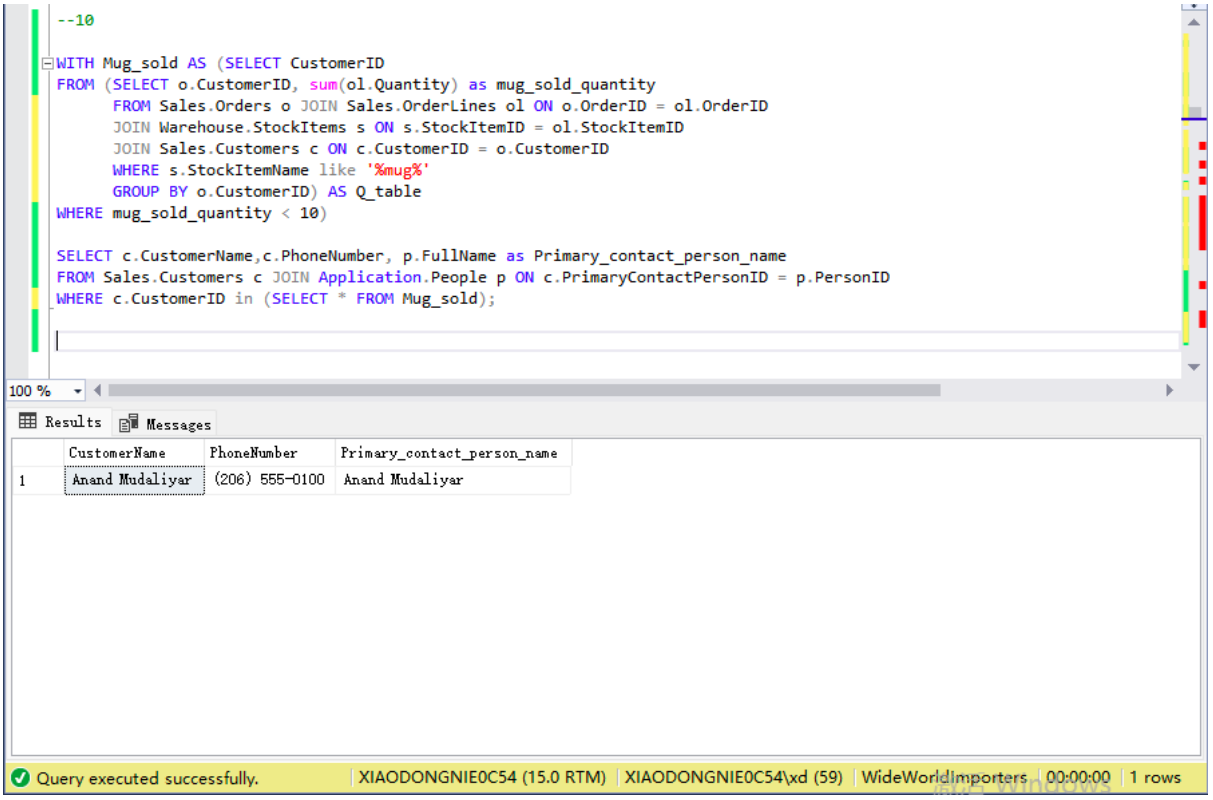
100 %

Results | Messages

	StockItemID
1	95
2	78
3	86
4	98
5	184
6	193
7	204
8	77
9	80

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 9 rows

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.



```
--10

WITH Mug_sold AS (SELECT CustomerID
FROM (SELECT o.CustomerID, sum(ol.Quantity) as mug_sold_quantity
      FROM Sales.Orders o JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID
      JOIN Warehouse.StockItems s ON s.StockItemID = ol.StockItemID
      JOIN Sales.Customers c ON c.CustomerID = o.CustomerID
      WHERE s.StockItemName like '%mug%'
      GROUP BY o.CustomerID) AS Q_table
WHERE mug_sold_quantity < 10)

SELECT c.CustomerName,c.PhoneNumber, p.FullName as Primary_contact_person_name
FROM Sales.Customers c JOIN Application.People p ON c.PrimaryContactPersonID = p.PersonID
WHERE c.CustomerID in (SELECT * FROM Mug_sold);
```

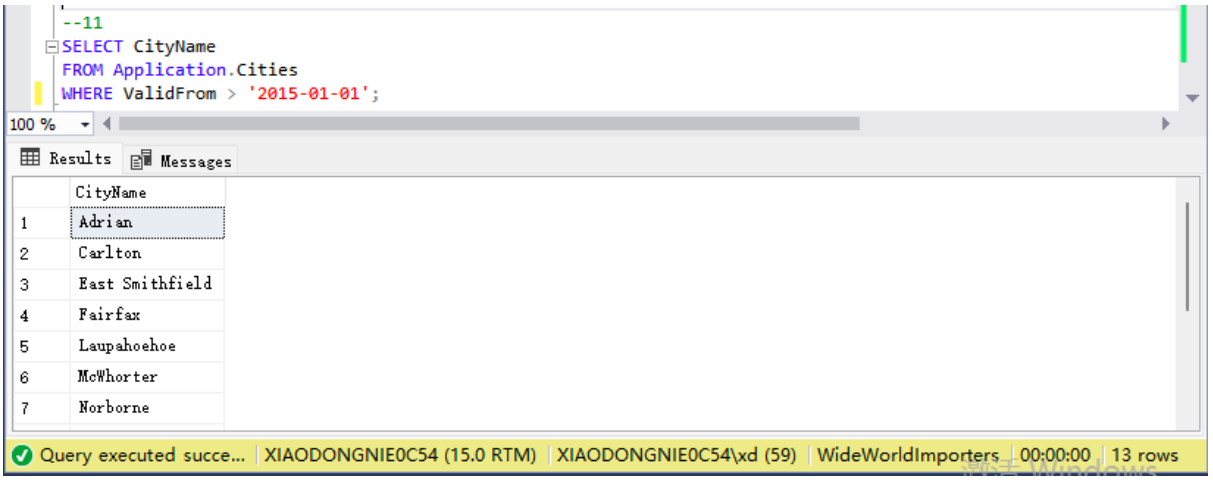
100 %

Results | Messages

	CustomerName	PhoneNumber	Primary_contact_person_name
1	Anand Mudaliyar	(206) 555-0100	Anand Mudaliyar

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 1 rows

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



```
--11
SELECT CityName
FROM Application.Cities
WHERE ValidFrom > '2015-01-01';
```

100 %

Results | Messages

	CityName
1	Adrian
2	Carlton
3	East Smithfield
4	Fairfax
5	Laupahoehoe
6	McWhorter
7	Norborne

Query executed succe... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 13 rows

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.

```sql
--12
SELECT s.StockItemName,c.DeliveryAddressLine1,c.DeliveryAddressLine2,
sp.StateProvinceName, ct.CityName,co.CountryID,c.CustomerName,p.FullName,c.PhoneNumber
FROM Sales.Orders o JOIN Sales.OrderLines o1 ON o.OrderID = o1.OrderID
JOIN Warehouse.StockItems s ON s.StockItemID = o1.StockItemID
JOIN Sales.Customers c ON o.CustomerID = c.CustomerID
JOIN Application.People p ON c.PrimaryContactPersonID = p.PersonID
JOIN Application.Cities ct ON ct.CityID = c.DeliveryCityID
JOIN Application.StateProvinces sp ON sp.StateProvinceID = ct.StateProvinceID
JOIN Application.Countries co ON co.CountryID = sp.CountryID
WHERE o.OrderDate = '2014-07-01'
```

100 %

Results | Messages

	StockItemName	DeliveryAddressLine1	DeliveryAddressLine2	StateProvinceName	CityName	CountryI
1	Red and white urgent despatch tape 48mmx75m	Unit 176	1674 Skujins Boulevard	New York	Gasport	230
2	Packing knife with metal insert blade (Yellow) 9mm	Unit 176	1674 Skujins Boulevard	New York	Gasport	230
3	Superhero action jacket (Blue) 3XS	Unit 176	1674 Skujins Boulevard	New York	Gasport	230
4	Animal with big feet slippers (Brown) S	Unit 176	1674 Skujins Boulevard	New York	Gasport	230
5	DBA joke mug - SELECT caffeine FROM mug (White)	Suite 185	1492 Shah Road	Illinois	Stonefort	230
6	Void fill 200 L bag (White) 200L	Suite 185	1492 Shah Road	Illinois	Stonefort	230
7	RC toy sedan car with remote control (Red) 1/50...	Suite 185	1492 Shah Road	Illinois	Stonefort	230
8	DBA joke mug - it depends (Black)	Suite 185	1492 Shah Road	Illinois	Stonefort	230
9	DBA joke mug - two types of DBAs (White)	Suite 185	1492 Shah Road	Illinois	Stonefort	230

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 180 rows

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

```sql
--13
WITH PurchaseTotal as (SELECT SUM(cast(p.ReceivedOuters * st.QuantityPerOuter as bigint)) as purchase_total,st.StockItemID
                       FROM Warehouse.StockItems st JOIN Purchasing.PurchaseOrderLines p ON p.StockItemID = st.StockItemID
                       GROUP BY st.StockItemID)

SELECT s2.StockGroupID, sum(p.purchase_total) as total_quantity_total, sum(o.Quantity) as total_quantity_sold,
sum(p.purchase_total)-sum(o.Quantity) as remaining_stock_quantity
FROM Warehouse.StockItems s1 LEFT JOIN Warehouse.StockItemStockGroups s2 ON s1.StockItemID = s2.StockItemID
JOIN PurchaseTotal p ON p.StockItemID = s1.StockItemID JOIN Sales.OrderLines o ON o.StockItemID = s1.StockItemID
GROUP BY s2.StockGroupID;
```

100 %

Results | Messages

	StockGroupID	total_quantity_total	total_quantity_sold	remaining_stock_quantity
1	1	6887658	1168276	5719382
2	2	92653646790	2842918	92650803872
3	3	1531215	244010	1287205
4	4	92650481544	1800072	92648681472
5	6	92653835893	2181299	92651654594
6	7	1669834	81057	1588777
7	8	1461466	395849	1065617
8	9	1379609	121360	1258249
9	10	49003686035	5838043	48997847992

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 9 rows

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”.

```
--14

WITH Max_city_item as (SELECT Cityid, MAX(Delivery_time) as Max_stock_item
                       FROM (SELECT ct.CityID, s.StockItemID, sum(o.OrderID) as Delivery_time
                             FROM Sales.Customers c JOIN Application.Cities ct ON c.DeliveryCityID = ct.CityID
                             JOIN Sales.Orders o ON c.CustomerID = o.CustomerID
                             JOIN Sales.OrderLines ol ON ol.OrderID = o.OrderID
                             JOIN Warehouse.StockItems s ON s.StockItemID = ol.StockItemID
                             WHERE YEAR(o.ExpectedDeliveryDate) = 2016
                             GROUP BY ct.CityID, s.StockItemID) as A
                      GROUP BY CityID)

SELECT c.CityID,c.CityName,
(CASE WHEN m.Max_stock_item IS NULL THEN 'NO SALES'
      ELSE str(m.Max_stock_item) END) AS City_max_stock_item
FROM Application.Cities c LEFT JOIN Max_city_item m ON c.CityID = m.CityID
ORDER BY m.Max_stock_item desc;
```

100 %

Results | Messages

	CityID	CityName	City_max_stock_item
1	18514	Lakemore	287790
2	16257	Huntington Woods	283457
3	21367	Maypearl	282173
4	1121	Argusville	281025
5	13896	Greenfield	278177
6	25170	Oconto Falls	277405
7	20189	Lucasville	275806
8	9064	Dickerson	274490
9	16702	Isabela	272277
10	15646	Hodgdon	263654

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:01 | 37,940 rows

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

```
--15
SELECT OrderID
FROM(SELECT OrderID, count(DeliveryMethodID) as delivery_time
     FROM Sales.Invoices
     GROUP by OrderID) AS A
where delivery_time > 1;
```

100 %

Results | Messages

	OrderID

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 0 rows

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

```sql
-- 16
SELECT StockItemID, StockItemName
FROM Warehouse.StockItems
WHERE CustomFields like '%China%';
```

100 %

Results | Messages

	StockItemID	StockItemName
1	1	USB missile launcher (Green)
2	2	USB rocket launcher (Gray)
3	3	Office cube periscope (Black)
4	16	DBA joke mug - mind if I join you? (White)
5	17	DBA joke mug - mind if I join you? (Black)
6	18	DBA joke mug - daaaaaa-ta (White)
7	19	DBA joke mug - daaaaaa-ta (Black)

Query executed su... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 207 rows

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

```sql
--17
SELECT Country_Of_Manufacture ,sum(Quantity) as Total_Quantity
FROM (SELECT ol.Quantity, s.StockItemID,
case when s.CustomFields like '%China%' then 'China'
     when s.CustomFields like '%Japan%' then 'Japan'
     else 'USA' END AS Country_Of_Manufacture
FROM Sales.OrderLines ol JOIN Sales.Orders o ON o.OrderID = ol.OrderID
JOIN Warehouse.StockItems s  ON s.StockItemID = ol.StockItemID
WHERE year(o.OrderDate) = 2015) AS New_table
GROUP BY Country_Of_Manufacture;
```

100 %

Results | Messages

	Country_Of_Manufacture	Total_Quantity
1	China	2850885
2	Japan	22365

Query executed succ... | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 2 rows

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]

```
--18
Create VIEW Stockgroup_year AS

WITH Stock_sold_year AS (SELECT StockGroupName, Sold_year,sum(Quantity) as Quantity_year
                         FROM (SELECT s3.StockGroupName,year(o.OrderDate) as Sold_year, ol.Quantity
                               FROM Sales.OrderLines ol JOIN Sales.Orders o ON ol.OrderID = o.OrderID
                               JOIN Warehouse.StockItems s ON ol.StockItemID = s.StockItemID
                               JOIN Warehouse.StockItemStockGroups s2 ON s2.StockItemID = s.StockItemID
                               JOIN Warehouse.StockGroups s3 ON s2.StockGroupID = s3.StockGroupID) AS new_table
                         GROUP BY Sold_year,StockGroupName)

SELECT *
FROM Stock_sold_year
PIVOT(
  SUM(Quantity_year) FOR Sold_year IN([2013],[2014],[2015],[2016])
) AS Stock_sold_year_pivot;

--Display the View
SELECT *
FROM Stockgroup_year;
```

100 %

Results | Messages

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

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 9 rows

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]

```
--19
Create VIEW Stock_year AS

WITH Stock_sold_year1 AS (SELECT StockGroupID, Sold_year,sum(Quantity) as Quantity_year
                          FROM (SELECT s3.StockGroupID,year(o.OrderDate) as Sold_year, ol.Quantity
                                FROM Sales.OrderLines ol JOIN Sales.Orders o ON ol.OrderID = o.OrderID
                                JOIN Warehouse.StockItems s ON ol.StockItemID = s.StockItemID
                                JOIN Warehouse.StockItemStockGroups s2 ON s2.StockItemID = s.StockItemID
                                JOIN Warehouse.StockGroups s3 ON s2.StockGroupID = s3.StockGroupID) AS new_table
                                GROUP BY Sold_year,StockGroupID),
Stock_pivot as  (SELECT *
FROM Stock_sold_year1
PIVOT(
  SUM(Quantity_year) FOR StockGroupID IN([1],[2],[3],[4],[5],[6],[7],[8],[9],[10])
) AS Stock_sold_year_pivot2)

SELECT Sold_year, "1" as 'Stock_Group_Name1',"2" as 'Stock_Group_Name2',"3" as 'Stock_Group_Name3',
"4" as 'Stock_Group_Name4',"5" as 'Stock_Group_Name5', "6" as 'Stock_Group_Name6',
"7" as 'Stock_Group_Name7',"8" as 'Stock_Group_Name8',"9" as 'Stock_Group_Name9',
"10" as 'Stock_Group_Name10'
FROM Stock_pivot;
-- Display the View
SELECT *
FROM Stock_year;
```

100 %

Results | Messages

	Sold_year	Stock_Group_Name1	Stock_Group_Name2	Stock_Group_Name3	Stock_Group_Name4	Stock_Group_Name5	Stock_Group_Name6	Stock_Group_Name7
1	2013	276609	767341	65713	486924	NULL	588555	21328
2	2014	306077	831573	70384	528096	NULL	639315	23685
3	2015	328677	889178	77268	558144	NULL	677480	26048
4	2016	256913	354826	30645	226908	NULL	275949	9996

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 4 rows

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

```
--20
DROP FUNCTION Total_price
CREATE FUNCTION Total_price(@id INT)
RETURNS INT
AS
BEGIN
    DECLARE @price int;
    SELECT @price = T_price
    FROM (SELECT o.OrderID, sum(ol.UnitPrice * ol.Quantity) AS T_price
    FROM Sales.Orders o JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID
    WHERE o.OrderID = @id
    GROUP BY o.OrderID) AS TP
    RETURN @price
END

SELECT *, ods.Total_price(OrderID) AS Totaol_order_price
FROM Sales.Invoices ;
```

100 %

Results | Messages

	InvoiceID	CustomerID	BillToCustomerID	OrderID	DeliveryMethodID	ContactPersonID	AccountsPersonID	SalespersonPersonID	PackedByPersonID
1	1	832	832	1	3	3032	3032	2	14
2	2	803	803	2	3	3003	3003	8	14
3	3	105	1	3	3	1209	1001	7	14
4	4	57	1	4	3	1113	1001	16	14

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:15 | 70,510 rows

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.

```sql
--21
--Create store procedure and new table

DROP PROCEDURE Orders_info
CREATE PROCEDURE Orders_info
@date date
--@OrderID int output,
--@OrderDate date output,
--@OrderTotal int output
--@CustomerID int output

AS  SET NOCOUNT ON;
    SELECT s.OrderID, s.CustomerID, s.OrderDate INTO ods.Orders --create a new table
    FROM Sales.Orders s
    WHERE s.OrderDate = @date;
GO

-- Run store producure and store data in new table
EXECUTE Orders_info '2013-01-02'

SELECT OrderID, CustomerID,OrderDate
FROM ods.Orders
```

100 %

Results | Messages

	OrderID	CustomerID	OrderDate
1	80	543	2013-01-02
2	81	456	2013-01-02
3	82	487	2013-01-02
4	83	154	2013-01-02
5	84	111	2013-01-02

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 74 rows

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.

```sql
--22

CREATE TABLE ods.StockItems(
    StockItemID INT PRIMARY KEY,
    StockItemName nvarchar(100) NOT NULL,
    SupplierID INT NOT NULL,
    ColorID INT NULL,
    UnitPackageID INT NOT NULL,
    OuterPackageID INT NOT NULL,
    Brand nvarchar(50) NULL,
    Size nvarchar(20) NULL,
    LeadTimeDays INT NOT NULL,
    QuantityPerOuter INT NOT NULL,
    IsChillerStock BIT NOT NULL,
    Barcode nvarchar(50) NULL,
    TaxRate DECIMAL(18, 3) NOT NULL,
    UnitPrice DECIMAL(18, 2) NOT NULL,
    RecommendedRetailPrice DECIMAL(18, 2) NULL,
    TypicalWeightPerUnit DECIMAL(18, 3) NOT NULL,
    MarketingComments nvarchar(MAX) NULL,
    InternalComments nvarchar(MAX) NULL,
    CountryOfManufacture nvarchar(20) NULL,
    [Range] nvarchar(20) NULL,
    Shelflife nvarchar(20) NULL
)
```

```
MERGE INTO ods.StockItems AS s1
USING Warehouse.StockItems AS s2
ON s1.StockItemID = s2.StockItemID
WHEN NOT MATCHED
THEN INSERT VALUES (s2.StockItemID,
                    s2.StockItemName,
                    s2.SupplierID,
                    s2.ColorID,
                    s2.UnitPackageID,
                    s2.OuterPackageID,
                    s2.Brand,
                    s2.Size,
                    s2.LeadTimeDays,
                    s2.QuantityPerOuter,
                    s2.IsChillerStock,
                    s2.Barcode,
                    s2.TaxRate,
                    s2.UnitPrice,
                    s2.RecommendedRetailPrice,
                    s2.TypicalWeightPerUnit,
                    s2.MarketingComments,
                    s2.InternalComments,
                    JSON_VALUE(s2.CustomFields, '$.CountryOfManufacture'),
                    JSON_VALUE(s2.CustomFields, '$.Range'),
                    JSON_VALUE(s2.CustomFields, '$.ShelfLife'));
```

100 %

Messages

```
(227 rows affected)

Completion time: 2022-11-04T02:11:13.6859407-07:00
```

100 %

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 0 rows

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.

```
--23
--If the procedure exist, drop it
DROP PROCEDURE Orders_day
-- Create procedure
CREATE PROCEDURE Orders_day
@date date
--@OrderID int output,
--@OrderDate date output,
--@CustomerID int output
AS  SET NOCOUNT ON;

    SELECT s.OrderID, s.CustomerID, s.OrderDate
    FROM Sales.Orders s
    WHERE datediff(day,@date,s.OrderDate) < 8 AND datediff(day,@date,s.OrderDate) > 0
GO

 -- Show the informtion of next 7 days of '2013-01-01'
EXECUTE Orders_day N'2013-01-01'
```

100 %

Results | Messages

	OrderID	CustomerID	OrderDate
1	80	543	2013-01-02
2	81	456	2013-01-02
3	82	487	2013-01-02
4	83	154	2013-01-02
5	84	111	2013-01-02

Query executed successfully. | XIAODONGNIE0C54 (15.0 RTM) | XIAODONGNIE0C54\xd (59) | WideWorldImporters | 00:00:00 | 379 rows

24

```
--24
DECLARE @json NVARCHAR(MAX);
SET @json = N'[
  {"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",
```

```
SELECT * INTO Json_table
FROM OPENJSON(@json)
  WITH (
    StockItemName nvarchar(MAX),
    SupplierID int '$.Supplier',
    UnitPackageID int '$.UnitPackageId',
    OuterPackageID int '$.OuterPackageId',
    Brand nvarchar(50),
    LeadTimeDays int,
    QuantityPerOuter int,
    TaxRate decimal(18,3),
    UnitPrice decimal(18,2),
    RecommendedRetailPrice decimal(18,2),
    TypicalWeightPerUnit decimal(18,3),
    CustomFileds nvarchar(max) '$.CountryOfManufacture',
    Range nvarchar(100),
    OrderDate datetime,
    DeliveryMethod nvarchar(100),
    ExpectedDeliveryDate datetime,
    SupplierReference nvarchar(100)
  )

-- In this Json file, it not contain informations about primary key column in three
-- tables(Stock Item, Pruchase Order and Order Lines). Therefore, the data from Json
-- file can not insert in to these three tables.

Insert into Sales.OrderLines(Sales.OrderLines.TaxRate)(select TaxRate From Json_table)
```

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

```
--25
SELECT *
FROM Stock_year
FOR JSON PATH
```

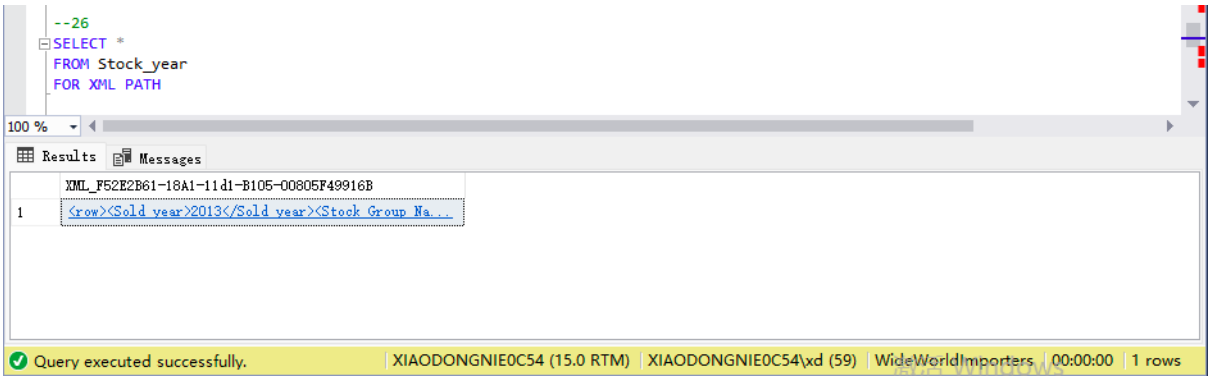
100 %

Results | Messages

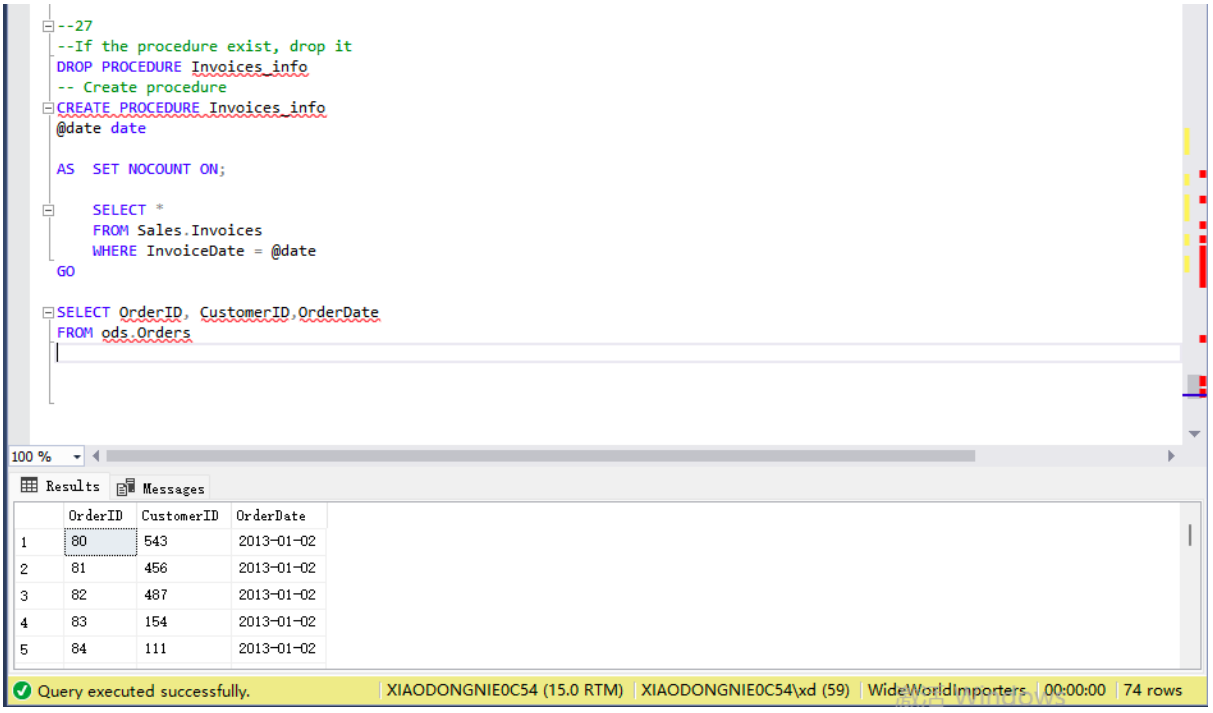
	JSON_F52E2B61-18A1-11d1-B105-00805F49916B
1	[{"Sold year":2013,"Stock Group Name1":276609,"...

Query executed successfully. XIAODONGNIE0C54 (15.0 RTM) XIAODONGNIE0C54\xd (59) WideWorldImporters 00:00:15 1 rows

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

The transaction is a grouping of one or more SQL statements. A transaction in its entire can commit to a database as a single logical unit or rollback as a single logical unit. In SQL, transactions are significant for maintaining database integrity. They are used to preserve integrity when multiple related operations are concurrently executed, or when multiple users interact with a database at same time. Therefore,

all of transaction should satisfied ACID principle. The acronym ACID stands for atomicity, consistency, isolation, and durability.

Atomicity : Atomicity refers to the fact that a transaction succeeds or it fails. The transaction composed of multiple steps, those steps are treated as a single operation or a unit. When a system stopped the database mid-transaction, the transaction fails, rolling the database back to the previous state.

Consistency: Consistency means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.

Isolation: DBMS allow users to access data concurrently and in parallel. Isolation is the characteristic that allows concurrency control so modifications from one transaction are not affecting operations in another transaction.

Durability: Durability refers to the persistence of committed transactions. Transactions and database modifications are not kept in
volatile memory but are saved to permanent storage. This prevents data loss during system failure.

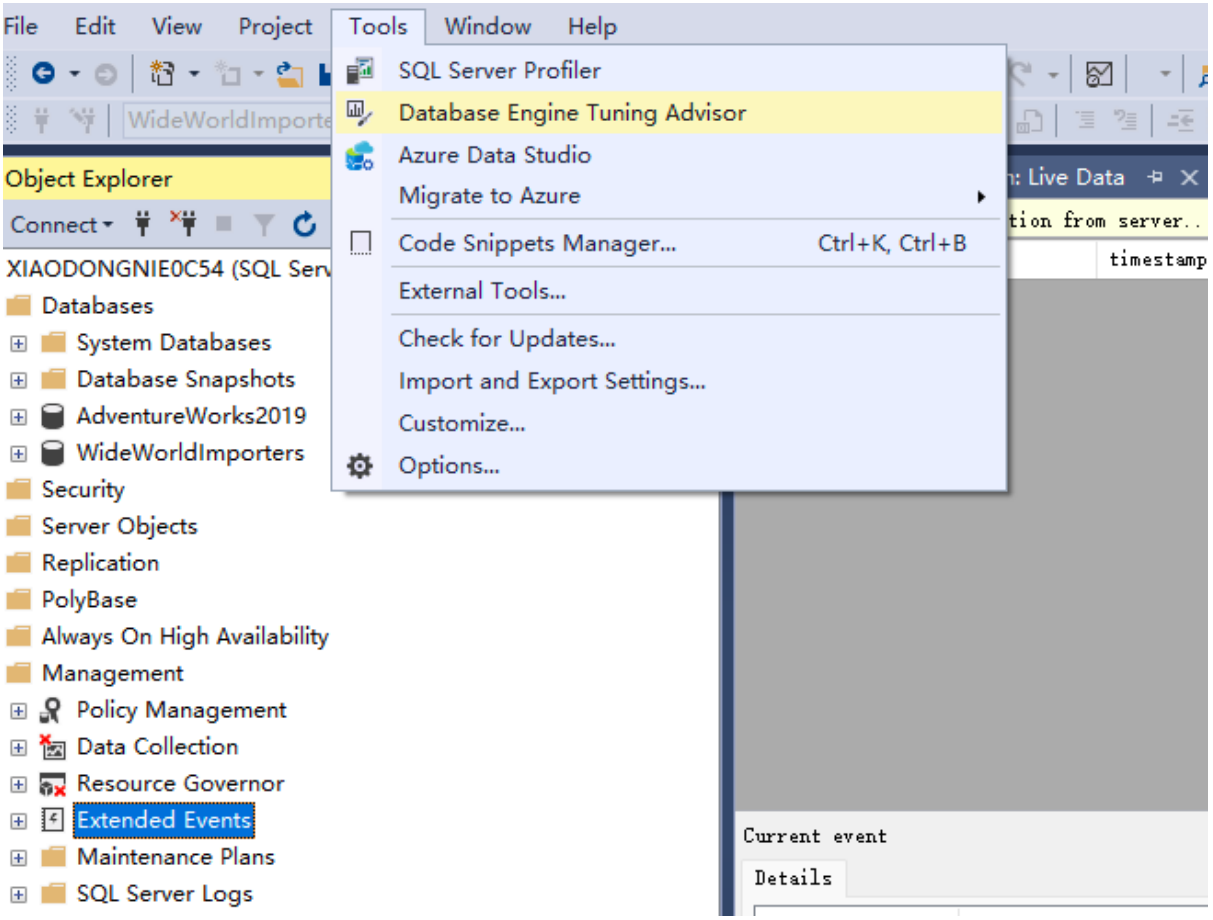
The SQL Server Transactions are classified into three types, they are Auto Commit Transaction Mode, Implicit Transaction Mode, Explicit Transaction Mode. Auto Commit Transaction Mode is the default transaction mode in SQL Server. And implicit transaction is when a new transaction is implicitly started when the prior transaction completes, but each transaction is explicitly completed with a commit or rollback statement. The last one is explicit transaction mode, is defined by the user that allows us to identify a transaction's beginning and ending points exactly. It will automatically abort in case of a error.

Transactions refer to an isolation level that defines how one transaction is isolated from other transactions. Isolation is the separation of resource or data modifications made by different transactions. Based on concurrency issues we have different levels of isolation levels. For example, snapshot, read uncommitted, read committed (system default), repeatable read, and serializable. The higher the isolation level, the more locking is involved. There is one common lock is deadlock. When two processes want to access tables that are mutually locked by each other, a deadlock occurs. In this condition, neither of the processes can move forward, as each process is waiting for the other process to release the lock resulting in a deadlock.

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.

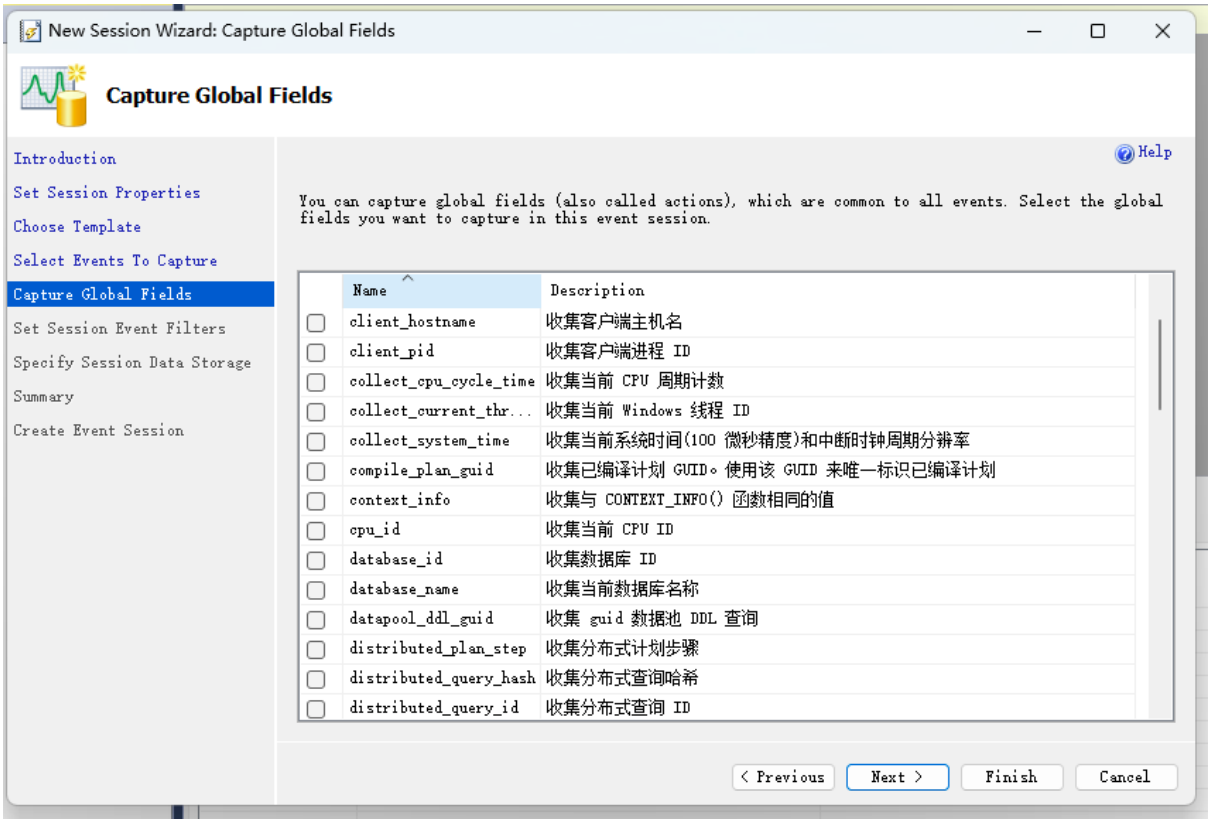
Tunning Advisor:

Tuning Advisor is a tool for analyzing workloads involved in database functioning. It enables the tuning of databases for improved query processing and the creation of an optimal set of indexes, indexed views and partitions. Tuning Advisor examines how queries are executed in a database and recommends methods to improve query processing.

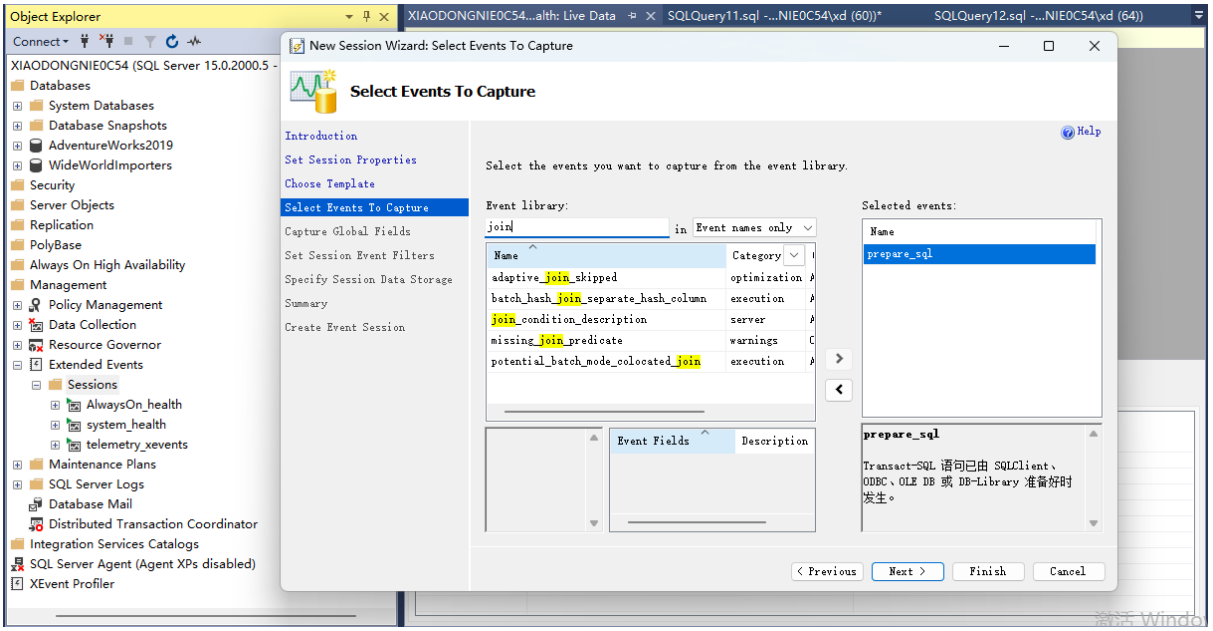


Extended Events:

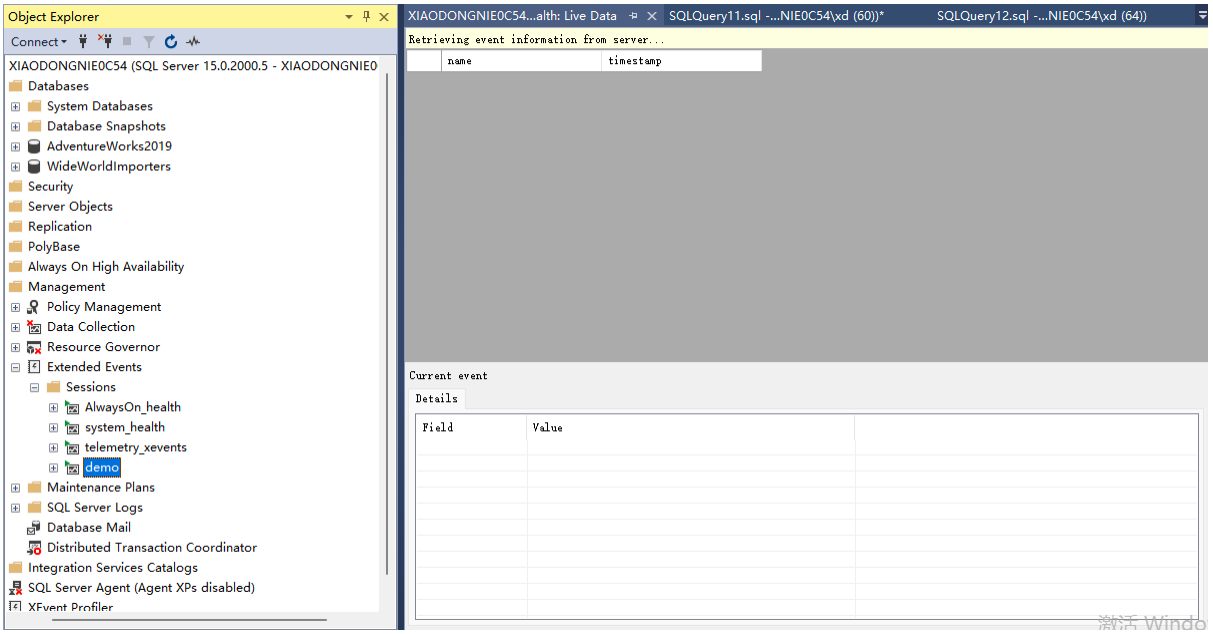
1 Create a new sessions and check the exist events.



2 Track the specific event that contain multiple fields



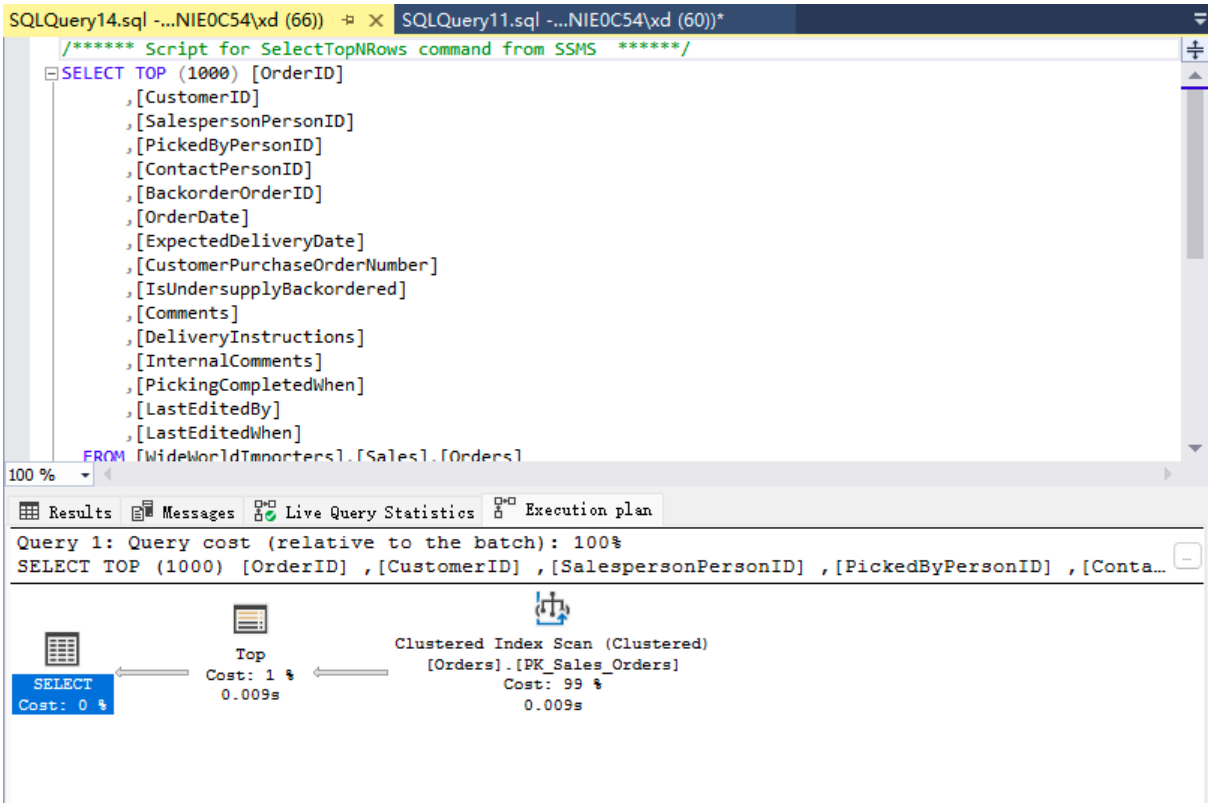
3 The session successfully create, use it to check events and fields.



DMV:
DMV is dynamic management views that can be used to monitor the health of a server instance, diagnose problems, and tune performance

Logs:
The Log File Viewer appears with a list of logs for you to view.

Execution Plan:



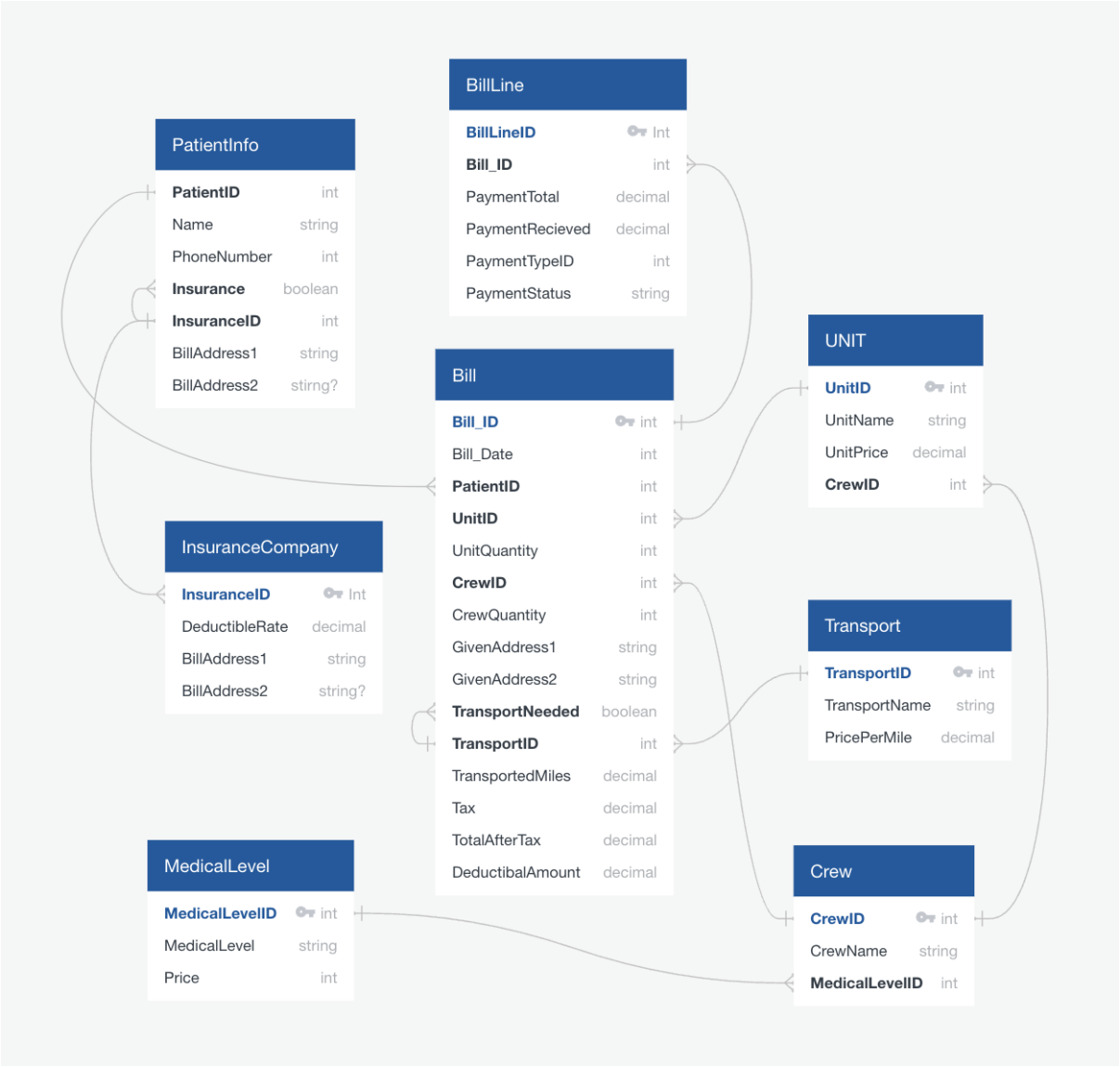
30 Write a short essay talking about a scenario: Good news everyone! We (Wide World Importers) just brought out a small company called “Adventure works”! Now that bike shop is our sub-company. The first thing of all works pending would be to merge the user logon information, person information (including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

```
IF OBJECT_ID(N'AdventureWorks2019.dbo.person', N'U') IS NOT NULL
  DROP TABLE AdventureWorks2019.dbo.person;
GO
CREATE TABLE AdventureWorks2019.dbo.person(
BusinessEntityID int not null,
FullName nvarchar (50) Not null,
PreferredName nvarchar (50) NOT NULL DEFAULT 'None',
SearchName nvarchar (101) NOT NULL DEFAULT 'None',
IsPermittedToLogon bit not null DEFAULT 0,
LogonName nvarchar (50),
IsExternalLogonProvider bit not null DEFAULT 0,
Hashedassword varbinary(max),
IsSystemUser bit not null DEFAULT 0,
IsEmployee bit not null DEFAULT 0,
IsSalesperson bit not null DEFAULT 0,
UserPreferences nvarchar(max),
PhoneNumber nvarchar (20),
FaxNumber nvarchar(20),
```

```
EmailAddress nvarchar(256),
Photo varbinary(max),
CustomFields nvarchar(max),
OtherLanguages nvarchar (max),
LastEditedby INT NOT NULL DEFAULT 0,
ValidFrom datetime2(7) NOT NULL DEFAULT GETDATE(),
ValidTo datetime2(7) NOT NULL DEFAULT '9999-12-31 23:59:59.9999999'
)
GO
--check table
select * from AdventureWorks2019.dbo.person
GO
--insert value
INSERT INTO AdventureWorks2019.dbo.person
(BusinessEntityID,FullName,EmailAddress,PhoneNumber)
SELECT p1.BusinessEntityID, CONCAT(p1.FirstName,' ',p1.LastName) as FullName,
p2.EmailAddress,p3.PhoneNumber
FROM Person.Person p1 JOIN Person.EmailAddress p2 ON p1.BusinessEntityID =
p2.BusinessEntityID
JOIN Person.PersonPhone p3 ON p1.BusinessEntityID = p3. BusinessEntityID
order by p1.BusinessEntityID
GO
--check table
select * from AdventureWorks2019.dbo.person
GO
--add personID to match with WWI
ALTER TABLE AdventureWorks2019.dbo.person
ADD PersonID int IDENTITY(3262,1)
GO
select * from AdventureWorks2019.dbo.person
GO
--alter personID datetype to match with WWI
ALTER TABLE AdventureWorks2019.dbo.person
ALTER COLUMN PersonID PRIMARY KEY
GO
--
INSERT INTO WideworldImporters.Application.People
select
PersonID,FullName,PreferredName,SearchName,IsPermittedToLogon,LogonName,IsExternalL
ogonProvider, Hashedassword, IsSystemUser, IsEmployee, IsSalesperson,
UserPreferences,PhoneNumber, FaxNumber, EmailAddress,Photo,CustomFields,
OtherLanguages, LastEditedby, ValidFrom, ValidTo
from AdventureWorks2019.dbo.person
```

Our team's code did not run successfully. We made a lot of progress but still stuck on the datatype converting. Attached below is the result code showing our understanding, progress and where we get stuck.

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:

1. Luckly, we only start with 1 fact: Purchase. Other facts can be ignored for now.
2. Add a new dimension: Country of Manufacture. It should be given on top of Stock Items.
3. W
   https://us04web.zoom.us/j/4736574443?pwd=b0FjU3ljVEduSkZJaUF1dTJUTkVlQT09rite script(s) and stored procedure(s) for the entire ETL from WWI db to DW.