SQL Assignments

SQL related assignments will be on the Wide World Importers Database unless otherwise mentioned.

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 AS Fax, P.PhoneNumber AS Phone, S.FaxNumber AS
   CompanyFax, S.PhoneNumber AS CompanyPhone
   FROM People P
   LEFT JOIN Suppliers S
   ON P.PersonID = S.PrimaryContactPersonID;
   ```
2. If the customer's primary contact person has the same phone number as the customer's phone number, list the customer companies.
   ```
   SELECT s.SupplierName
   FROM Suppliers s
   JOIN Customers c
   ON s.PhoneNumber = c. s.PhoneNumber
   GROUP BY s.PrimaryContactPersonID;
   ```
3. List of customers to whom we made a sale prior to 2016 but no sale since 2016-01-01.
   ```
   SELECT c.CustomerName
   FROM Customers c
   JOIN Orders o
   ON c.CustomerID = o.CustomerID
   WHERE o.OrderDate < 2016-01-01;
   ```
4. List of Stock Items and total quantity for each stock item in Purchase Orders in Year 2013.
   ```
   SELECT s.StockItemName, SUM(ol.Quantity) AS total_ quantity
   FROM StockItems s
   JOIN OrderLines ol
   ON s.StockItemID = ol.StockItemID
   WHERE YEAR(ol.PickingCompletedWhen) = 2016
   GROUP BY s.StockItemName;
   ```
5. List of stock items that have at least 10 characters in description.
   ```
   SELECT StockItemName
   FROM (
   SELECT StockItemName, LEN(StockItemName) AS length
   FROM StockItems)
   WHERE length >10;
   ```
6. List of stock items that are not sold to the state of Alabama and Georgia in 2014.
   ```
   SELECT StockItemName
   FROM StockItems si
   JOIN StateProvinces sp
   ON si. StockItemID = sp. StockItemID
   ```

```
WHERE StateProvinceName != ‘Alabama’ OR StateProvinceName != ‘Georgia’
AND YEAR(si.PickingCompletedWhen) = 2014;
```

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

```
SELECT sp.StateProvinceName, AVG(DATE(ol.Order_Date))
FROM StateProvince sp
JOIN OrderLines ol
ON sp.StateProvinceID = ol.StateProvinceID
GROUP BY sp.StateProvinceName;
```

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

```
SELECT sp.StateProvinceName, AVG(MONTH(ol.Order_Date))
FROM StateProvince sp
JOIN OrderLines ol
ON sp.StateProvinceID = ol.StateProvinceID
GROUP BY sp.StateProvinceName;
```

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

```
WITH CTE AS(
SELECT s.StockItemName,
COUNT(p.PurchaseOrderID) AS cnt_purchase,
COUNT(o.OrderID) AS cnt_sold AS cnt_sold
FROM StockItems s
JOIN PurchaseOrderLines p
ON s. StockItemID  = p.StockItemID
JOIN OrderLines o
ON s. StockItemID =o. StockItemID)

SELECT StockItemName
FROM CTE
WHERE YEAR(o.PickingCompletedWhen) = 2015 AND cnt_purchase >
cnt_sold;
```

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.

```
SEELCT c.CustomerName, c.PhoneNumber, p.FullName
FROM Customers c
JOIN Person p
ON c.PrimaryContactPersonId = p.PersonID
JOIN Order o
ON c.CustomerID = o. CustomerID
WHERE YEAR(o.OrderDate) = 2016
GROUP BY c.CustomerName
HAVING COUNT(o.OrderID) > 10;
```

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

```
SELECT c.CityName
FROM Cities c
```

```
JOIN StateProvinces sp
ON c.StateProvinceID = sp.StateProvinceID
JOIN Countries co
ON sp.CountryID = co. CountryID
WHERE DATE(co.ValidFrom) = ‘2015-01-01’;
```

12. List all the Order Detail (Stock Item name, delivery address, sp.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, s.DeliveryAddressLine1 AS delivery_address,
sp.StateProvinecDate AS delivery_state, c.CityName, co.CountryName,
cust.Customer_Name, p.FullName AS customer_contact_person_name,
cust_PhoneNumber AS customer_phone, ol.quantity
FROM StockItems si
JOIN Supplier s
ON si.SupperlierID = s. SupperlierID
JOIN Customer cust
ON s.PrimaryContactPersonID = cust.PrimaryContactPersonID
JOIN People p
ON cust.PrimaryContactPersonID = p.PersonID
JOIN Orders o
ON p.PersonID = o. PickedByPersonID
JOIN OrderLines ol
ON o,OrderID = ol.OrderID
JOIN Cities c
ON c.CityID = s.DeliveryCityID
JOIN StateProvinces sp
ON c.StateProvinceID =sp. StateProvinceID
JOIN Countries co
ON co.CountryID = sp.CountryID
GROUP BY SELECT si.StockItemName
WHERE DATE(ol.PickingCompletedWhen) = ‘2014-07-01’;
```

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

```
SELECT si.StockItemName, ol.quantity, SUM(ol.quantity) AS sum_quality
FROM StockItems si
JOIN Supplier s
ON si.SupperlierID = s. SupperlierID
JOIN Customer cust
ON s.PrimaryContactPersonID = cust.PrimaryContactPersonID
JOIN People p
ON cust.PrimaryContactPersonID = p.PersonID
JOIN Orders o
ON p.PersonID = o. PickedByPersonID
JOIN OrderLines ol
ON o,OrderID = ol.OrderID
JOIN Cities c
```

```
ON c.CityID = s.DeliveryCityID
JOIN StateProvinces sp
ON c.StateProvinceID =sp. StateProvinceID
JOIN Countries co
ON co.CountryID = sp.CountryID
GROUP BY SELECT si.StockItemName;
```

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”.

```
SELECT CityName, StockItemName
FROM(
SELECT c.CityName, si.StockItemName, CASE WHEN(c.PurchaseID = 0
THEN ‘No Sales’) AS sales
FROM StockItems si
JOIN Supplier s
ON si.SupperlierID = s. SupperlierID
JOIN Customer cust
ON s.PrimaryContactPersonID = cust.PrimaryContactPersonID
JOIN People p
ON cust.PrimaryContactPersonID = p.PersonID
JOIN Orders o
ON p.PersonID = o. PickedByPersonID
JOIN OrderLines ol
ON o,OrderID = ol.OrderID
JOIN Cities c
ON c.CityID = s.DeliveryCityID
JOIN StateProvinces sp
ON c.StateProvinceID =sp. StateProvinceID
JOIN Countries co
ON co.CountryID = sp.CountryID
WHERE YEAR(ol.PickingCompletedWhen) = 2016
GROUP BY c.CityName);
```

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

```
SELECT OrderID
FROM (
SELECT o.OrderID, COUNT(i.DeliveryMethodID) AS cnt
FROM Orders o
JOIN Invoices i
ON i.OrderID =o.OrderID
GROUP BY o.OrderID)
WHERE cnt >1;
```

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

```
SELECT StockItemName
FROM StockItems
WHERE CustomFilds LIKE ‘%China%’;
```

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

```
SELECT si.StockItemName, SUM(sh.LastStocktakeQuantity) AS sum_quality
```

```
FROM StockItems si
JOIN StockItemHoldings sh
ON si. StockItemID = sh.StockItem
WHERE YEAR(sh. LastEditedWhen) = 2015
GROUP BY sp.CountryID;
```

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 table_sample
AS SELECT StockItemName,
SUM(QuantityPerOuter) AS sum_quan,
CASE WHEN(YEAR(OrderDate)=2013 THEN 2013
        WHEN(YEAR(OrderDate)=2014 THEN 2014
        WHEN(YEAR(OrderDate)=2015 THEN 2015
        WHEN(YEAR(OrderDate)=2016 THEN 2016
        WHEN(YEAR(OrderDate)=2017 THEN 2017
        ELSE NULL END)
FROM StockItems, orders
WHERE StockItem. StockItemID = orders.StockItemID
GROUP BY StockItemName;
```

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 table_sample
AS SELECT sgn.StockGroupsName,
SUM(o.QuantityPerOuter) AS sum_quan,
CASE WHEN(o.YEAR(OrderDate)=2013 THEN 2013
        WHEN(o.YEAR(OrderDate)=2014 THEN 2014
        WHEN(o.YEAR(OrderDate)=2015 THEN 2015
        WHEN(o.YEAR(OrderDate)=2016 THEN 2016
        WHEN(o.YEAR(OrderDate)=2017 THEN 2017
        ELSE NULL END)
FROM StockGroups sg
JOIN StockItemStockGroups sisg
ON sg.StockGroupID = sisg. StockGroupID
JOIN StockItems si
ON sisg.StockItemID = si. StockItemID
WHERE YEAR(OrderDate) BETWEEN 2013 AND 2017
GROUP BY StockItemName;
```

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 OR ALTER FUNCTION order_information
( [ { @ OrderID order_id int
 [ = default ] [ READONLY ] }
  ]
)
RETURNS sum_quantity int
  BEGIN
    function_body
    RETURN sum_quantity
  END
;
```

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.

```
--Step 1: create a new table
create or replace ods.Orders
(order_id int
order_date date,
order_total int,
customer_id int);

--Step 2: create a stored procedure
create or replace procedure ods.Orders.sp_tbl_bkp_PantingZhao (Date date,
dt_interval number)
returns table()
language sql
as
$$
declare
   fu_date varchar;
   cur_date varchar;
   bkp_name varchar;
   select_table varchar;
   rs resultset;
begin
   fu_date := replace(dateadd(day, 7, current_date()), '-', '');
   cur_date := replace(current_date(), '-', '');
   bkp_name := concat('hackerrank.playground_dev.DEL_', fu_date, '_BKP_',
tbl_nm, '_PANTINGZHAO_', cur_date);
   select_table := concat('create or replace table ', bkp_name , ' as select * from ',
tbl_nm);
   rs := (execute immediate :select_table);
   return table(rs);
end;
$$
;
```

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 OR REPLACE TABLE ods.StockItem clone
SELECT *
FROM WideWorldImporters.Warehouse.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.

```
create or replace procedure ods.Orders.sp_tbl_bkp_PantingZhao (Date date,
dt_interval number)
returns table()
language sql
as
$$
declare
    fu_date varchar;
    cur_date varchar;
    bkp_name varchar;
    select_table varchar;
    rs resultset;
begin
    fu_date := replace(dateadd(day, 7, current_date()), '-', '');
    cur_date := replace(current_date(), '-', '');
    bkp_name := concat('hackerrank.playground_dev.DEL_', fu_date, '_BKP_',
tbl_nm, '_PANTINGZHAO_', cur_date);
    select_table := concat('create or replace table ', bkp_name , ' as select * from ',
tbl_nm);
    rs := (execute immediate :select_table);
    return table(rs);
end;
$$
;
```

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.

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

```
CREATE OR REPLACE TABLE table_sample
    AS SELECT sgn.StockGroupsName,
SUM(o.QuantityPerOuter) AS sum_quan,
CASE WHEN(o.YEAR(OrderDate)=2013 THEN 2013
          WHEN(o.YEAR(OrderDate)=2014 THEN 2014
          WHEN(o.YEAR(OrderDate)=2015 THEN 2015
          WHEN(o.YEAR(OrderDate)=2016 THEN 2016
          WHEN(o.YEAR(OrderDate)=2017 THEN 2017
          ELSE NULL END)
FROM StockGroups sg
JOIN StockItemStockGroups sisg
ON sg.StockGroupID = sisg. StockGroupID
JOIN StockItems si
ON sisg.StockItemID = si. StockItemID
WHERE YEAR(OrderDate) BETWEEN 2013 AND 2017
GROUP BY StockItemName
For JSON PATH, ROOT('StockGroups', 'StockItemStockGroups', 'StockItems');
```

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

```
CREATE VIEW table_sample
AS SELECT sgn.StockGroupsName,
SUM(o.QuantityPerOuter) AS sum_quan,
CASE WHEN(o.YEAR(OrderDate)=2013 THEN 2013
          WHEN(o.YEAR(OrderDate)=2014 THEN 2014
          WHEN(o.YEAR(OrderDate)=2015 THEN 2015
          WHEN(o.YEAR(OrderDate)=2016 THEN 2016
          WHEN(o.YEAR(OrderDate)=2017 THEN 2017
          ELSE NULL END)
FROM StockGroups sg
JOIN StockItemStockGroups sisg
ON sg.StockGroupID = sisg. StockGroupID
JOIN StockItems si
ON sisg.StockItemID = si. StockItemID
WHERE YEAR(OrderDate) BETWEEN 2013 AND 2017
GROUP BY StockItemName
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.

```
--Step 1: create a new table

create or replace ods.ConfirmedDeviveryJson

(id int
```

date date,

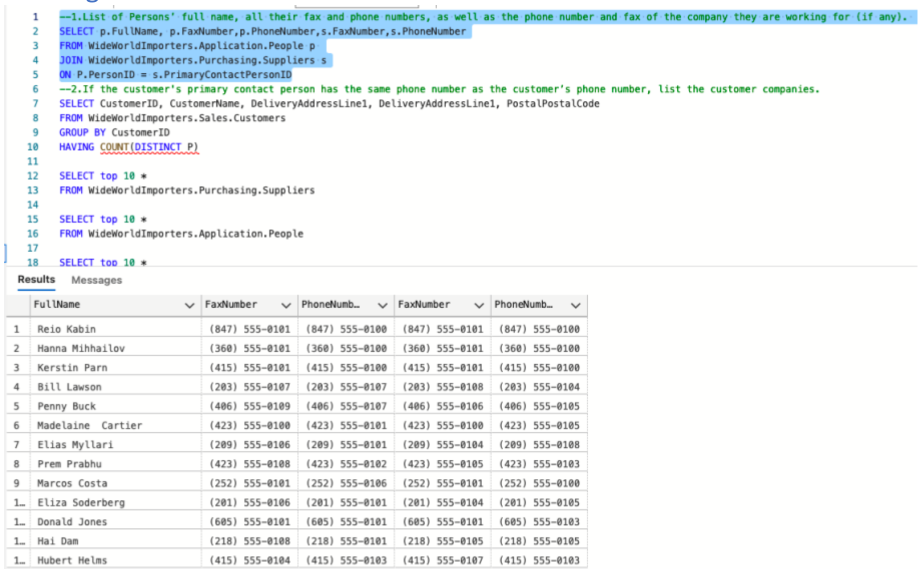
value int,

);

--Step 2: create a stored procedure

create or replace procedure ods.Invoices.sp_tbl_bkp_PantingZhao (date date, dt_interval number

28. Write a short essay talking about your understanding of transactions, locks and isolation levels.
    Transactions specify an isolation level that defines how one transaction is isolated from other transactions.
    A database lock is used to "lock" some data in a database so that only one database user/session may update that particular data. So, database locks exist to prevent two or more database users from updating the same exact piece of data at the same exact time.
    Isolation is the separation of resource or data modifications made by different transactions. Isolation levels are described for which concurrency side effects are allowed, such as dirty reads or phantom reads.
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.
    I would like to take one query example to explain these.
    Firstly, here is the query that I wrote to output the Persons' full name, all their fax and phone numbers, as well as the phone number and fax of the company they are working for.

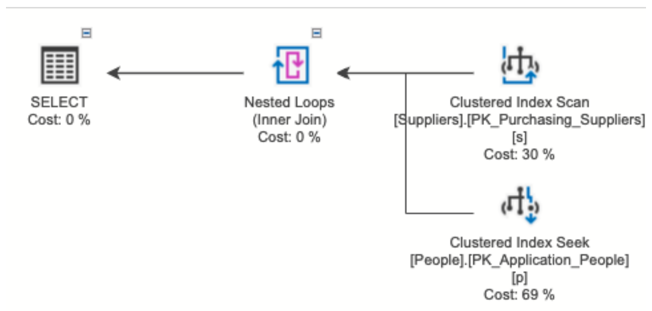


```
--1.List of Persons' full name, all their fax and phone numbers, as well as the phone number and fax of the company they are working for (if any).
SELECT p.FullName, p.FaxNumber,p.PhoneNumber,s.FaxNumber,s.PhoneNumber
FROM WideWorldImporters.Application.People p
JOIN WideWorldImporters.Purchasing.Suppliers s
ON P.PersonID = s.PrimaryContactPersonID
--2.If the customer's primary contact person has the same phone number as the customer's phone number, list the customer companies.
SELECT CustomerID, CustomerName, DeliveryAddressLine1, DeliveryAddressLine1, PostalPostalCode
FROM WideWorldImporters.Sales.Customers
GROUP BY CustomerID
HAVING COUNT(DISTINCT P)

SELECT top 10 *
FROM WideWorldImporters.Purchasing.Suppliers

SELECT top 10 *
FROM WideWorldImporters.Application.People

SELECT top 10 *
```

Results Messages

	FullName	FaxNumber	PhoneNumb…	FaxNumber	PhoneNumb…
1	Reio Kabin	(847) 555-0101	(847) 555-0100	(847) 555-0101	(847) 555-0100
2	Hanna Mihhailov	(360) 555-0101	(360) 555-0100	(360) 555-0101	(360) 555-0100
3	Kerstin Parn	(415) 555-0101	(415) 555-0100	(415) 555-0101	(415) 555-0100
4	Bill Lawson	(203) 555-0107	(203) 555-0107	(203) 555-0108	(203) 555-0104
5	Penny Buck	(406) 555-0109	(406) 555-0107	(406) 555-0106	(406) 555-0105
6	Madelaine Cartier	(423) 555-0100	(423) 555-0101	(423) 555-0100	(423) 555-0105
7	Elias Myllari	(209) 555-0106	(209) 555-0101	(209) 555-0104	(209) 555-0108
8	Prem Prabhu	(423) 555-0108	(423) 555-0102	(423) 555-0105	(423) 555-0103
9	Marcos Costa	(252) 555-0101	(252) 555-0106	(252) 555-0101	(252) 555-0100
1…	Eliza Soderberg	(201) 555-0106	(201) 555-0101	(201) 555-0104	(201) 555-0105
1…	Donald Jones	(605) 555-0101	(605) 555-0101	(605) 555-0101	(605) 555-0103
1…	Hai Dam	(218) 555-0108	(218) 555-0101	(218) 555-0105	(218) 555-0105
1…	Hubert Helms	(415) 555-0104	(415) 555-0103	(415) 555-0107	(415) 555-0103

From this screenshot, we can see the output tables including related information.



Then I check the execution plan to check the query performance.
We can check this plan map from right to left, which shows clustered Index Scan for the primary key in Suppliers table cost 30% and Clustered Index Seek in People cost 69%. Also, because of using inner join, it does not cost more time. So, from this execution plan, this query is fast enough to run this query.

Assignments 30 - 32 are group assignments.
For the group assignment, I just joined this Azure DE batch this Wednesday, so I've not been divided into groups. I will pick 1 question and try to figure it out.

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.

I want to share my idea according to question 1:

- Step 1: Create a new_table which includes the same column names with Person table in Wide World Importers Datebase
- Step 2: Use INSERT OVERWRITE INTO new_table to merge all the information together.

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: Purchase. 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.