Question 1-

As part of a recurring promotion, AdventureWorks would like to send an email each month to individual customers whose number of years between the date of their first purchase at AdventureWorks and the date of their last purchase is greater than or equal to one digit arbitrary (e.g. 10+, 15+) etc. AdventureWorks is a company that has been around for a long time now and their customers have been shopping with them long before the transaction system that is used now and for which you have access to the database.

The information recording the date of the first purchase of each customer is thus kept inside an XML field stored in the Person.Person table and called Demographics. It is in this field that you will have to retrieve the date of the first purchase of each individual customer in order to compare it to the date of the last purchase which you can easily find in the AdventureWorks sales table as you are used to.

We therefore need a function with the following specifications:

- The function name will be fn_GetLoyalCustomers
- The code must create or alter the function autonomously (i.e. no need to delete the function manually to recreate it)
- A smallint type input parameter that represents a minimum number of years. Call this parameter @aNumberOfYears
- The function must return a table with the following information: Person identifier (individual customer), first name, last name, Date of first purchase (see above), date of last purchase, number of years between the two dates
- We will only return the information of customers whose number of years specified in parameter is greater than or equal to that passed in parameter in the function.

To do this, consider the following information:

(1) Data extraction from XML type fields in SQL Server is not done through text extraction. This is a structured data type with an XML schema as a reference. Tip: before worrying about developing the function, work on extracting the information requested in the XML field. After, integrate this into the complete query, and finally, wrap it all in the function, it will make your job a lot easier. The SQL Server documentation provides various information and examples related to extracting XML fields. There are also many resources online. Remember here you want to fetch a value from an XML type attribute.

(2) You can assume that all individual customers provide the requested information. There is no NULL value to anticipate.

(3) To test your function, you should be able to run the following query: SELECT * FROM fn_GetLoyalCustomers(10); And thus obtain the list of customers whose number of years between the date of their first order and their last order is greater than or equal to 10.

********** QUESTION 2 **********

The shipping and receiving department needs your help to help them redefine the way they calculate shipping costs for orders to be sent. The idea is this: Every person who has a case with AdventureWorks has at least one shipping address, as listed in the Customer Orders SalesOrderHeader table.

In the address we find a bunch of interesting information but what is particularly interesting here is the information contained in the SpatialLocation attribute of the Person.Address table. Unluckily, when we consult this attribute, SQL Server seems to return everything except location information. This is because it is a "geography" type field, which is a specific type used for geographical representations (more specifically it is a so-called "CLR" type which comes from the .NET language. It You will therefore have to learn how to work with this type of attribute here.

To start, run a simple query like:

SELECT TOP(10) AddressID, SpatialLocation FROM Person.Address; SpatialLocation offers methods (watch out for capitals!) like ToString():

SELECT TOP(10) AddressID, SpatialLocation, SpatialLocation.ToString() FROM Person.Address;

We can thus see that SpatialLocation offers a longitude and a latitude. They can also be viewed as:

```
SELECT TOP(10)
                        AddressID
                        , SpatialLocation
                        , SpatialLocation.ToString()
                        , SpatialLocation.Long
                        , SpatialLocation.Lat
                FROM Person.Address;
```

With this information, we see that we can know the exact geolocation of an address directly from SQL Server (cool!).

In the exercise requested here, we would like to calculate the distance between AdventureWorks HQ and each of the addresses for shipping orders at AdventureWorks. Obviously, in theory, we should calculate this distance in terms of roads etc. like Google Maps would.

That said, to make it simpler, we will calculate a so-called geodesic distance between two points: the AdventureWorks HQ, and the shipping address of an order.

Here are the steps required to perform this exercise:

(1) Declare a variable (@HQLocation) to store the coordinates of AventureWorks HQ. The HQ is located at Building 9, 1 Microsoft Way Redmond WA. By typing this address on Google Maps, you can copy the coordinates into your SQL window.

Now you have to store this information in your variable, which is obviously of Geography type like SpatialLocation in the address table. To do this, see the SQL Server documentation for creating a geographic "point": https://docs.microsoft.com/en-us/sql/t-sql/spatial-geography/point-geography-data-type? view=sql-server-ver16

Congratulations, after that you should have a variable that stores the address of AdventureWorks HQ (which oddly coincides with Microsoft HQ...).

(2) Write a separate query that returns the following information: Address ID, first line of address, city, province/state name, country name, address latitude, address longitude . All for shipping addresses for orders at AdventureWorks.

(3) Add a new column to your query written in (2). This is the distance between the coordinates of the HQ address and each of the addresses.

To do this, see the documentation on distance calculation in SQL Server: https://docs.microsoft.com/en-us/sql/t-sql/spatial-geography/stdistance-geography-data-type?view=sql -server-ver16

you now have a nice query that returns almost the requested information, well done! Note that the distance returned is in meters. You can for example check that roughly speaking, the distances you have correspond to those that we would have more or less as the crow flies (there are technical details here which we do without but basically, it remains an approximate measurement of distance which is quite close to a true geodetic measurement).

(4) Now you need to "package" your request into a function called fn_GetShippingTiers() which takes no parameters. The function returns the previous information with two differences. First of all, we now want to express the distance in kilometers, to 2 decimal places. The calculation doesn't really change, it's more formatting. Second, we want to display a third related to the calculated distance by displaying the following information:

```
Distance entre 0 et 25: 'Tiers 1'
                              Distance entre 26 et 100: 'Tiers 2'
                              Distance entre 101 et 500: 'Tiers 3'
                              Distance entre 501 et 5000: 'Tiers 4'
                              Distance entre 5001 et 10000: 'Tiers 5'
                              Sinon 'Tiers X'
```