AdventureWorks – Solving Business Problems

André Assis dos Santos Gonçalves Loureiro m20201044@novaims.unl.pt
Gonçalo Daniel Tavares Duarte m20201329@novaims.unl.pt
Hugo Leandro Fortunato Gonçalves m20201785@novaims.unl.pt

***Abstract* – this paper presents methods to solve two business problems faced by AdventureWorks company. These methods include: the extension of company's database schema, in order to fulfill a new marketing campaign; the creation of specific views for financial report production; and business analysis to support a certain business decision. We used T-SQL to build all the required solutions, while managing relation data.**

Keywords – T-SQL; schema; views; stored procedures; managing relational data.

I. Introduction

AdventureWorks is a fictional company created by Microsoft in 2014. Its core business consists of manufacturing bicycles and some of its components. Their product line includes 97 different brands of bikes, grouped into three main categories: mountain, road, and touring. However, it also resells other bicycle components, accessories, and clothing, which are bought from over 100 suppliers.

The company's customer base can be divided into two categories: retail stores (over 700) and individuals (over 19.000), which are spread all over the United States, Canada, Australia, United Kingdom, France and Germany.

AdventureWorks does not have any physical stores, so its revenue comes from individual customers online shopping and retail stores bulk purchases, through company's sales team. Its employees' structure is composed by 290 people, covering sales, production, purchasing, engineering, finance, information services, marketing, logistics, and R&D.

Currently, AdventureWorks has two business challenges, that we will try to solve along this report.

II. Methods

Through this chapter we will explain the methods applied, in order to solve the different business problems imposed by AdventureWorks.

i. Stock clearance

Stock clearance is a business problem that AdventureWorks faces every year. Due to a considerable stock of previous bicycle models, the company has not been able to effectively replace them with the new ones. For this reason, AdventureWorks' leadership has decided to launch an online auction, to try to clear the existing stock. In order to implement this campaign, we will need to extend AdventureWorks database model, so it is ready to integrate the new required features. Since this campaign is going to occur over the last 2 weeks of December, including Black Friday, we will need to ensure website reliability, due to expected requests overload.

ii. Stock clearance – Schema

In order to prepare for the online auction campaign, we have extended AdventureWorks database with a new schema named Auctions. This schema is composed by three new tables, designed to support the workflow around the stock clearance:

- **Configuration Table:** This table contains necessary parameters to control the auction and gives the ability to change if needed. The parameters that can be adjusted are: minimal bid increment value, minimum price for enlistment in auction, maximum bid in percentage of list price that each customer can place, the start and end date of the auction, the initial price for in-house and supplied products (handled separately) defined as a percentage of the listed price of the product, and the default expiry date of the auction for each product.
- **Auctions Table:** This table stores the relevant information for each auctioned product. It was decided to give each new auctioned product an **Auction ID** as the **Primary Key**. **Product ID** is set as a **Foreign Key** linked to the other schemas and tables of the Adventureworks. A column for the initial price of each product and the maximum allowed bid is stored in the table. The current highest bidding price on the product was added since it is relevant information about

the auction status and is useful to control the workflow of the **TryBidProduct SP**. Besides the start and expire data of the product in the auction, it was decided to add 2 columns for the **Canceled Date**, to know the exact moment the product was removed from the auction, and **Sold Date**, which is attributed a value when the maximum bid is reached by a customer or when the auction ends. The **Status** column, together with the 4 date columns are defined to manage the **UpdateProductAuctionStatus SP.**

- **Bids Table:** This table keeps track of all bids for each product for all customers. The **BidID** is set as a **Primary Key**. **Customer ID** and **Auction ID** are set as **Foreign Keys**. For each bid it is stored the bid amount and the date.

iii. Stock clearance – Stored Procedures

With the new schema, we were able to create new stored procedures (SP) to fulfill the needs of the auction campaign.

- **UspAddProductToAuction –** This SP adds products to auction. Only current commercialized products, which are not accessories, not auctioned and cost more than 50$ can be included in the auction. This SP contains three parameters: @ProductID, @ExpireDate and @InitialBidPrice. The minimum initial bid price, unless specified, should correspond to 75% of list price, if the products are not manufactured in-house, and 50% for the rest. The maximum bid is defined as 100% of the list price of the product as default. It is assumed that products can only be added to the auction before it starts, nonetheless the date of insertion is registered. The expire date of the product on the auction is registered but is only considered if it doesn't exceed the stop bid date defined in the configuration table for the auction. If the expire date is not defined, 1 week is added to the start bid date defined in the configuration table. Status of the product is set to 1 representing an active product in the auction. It is assumed that a product can only be added to the auction one time.
    - Tables used: Auctions.AuctionProducts, Auction.Config, Production.Product, ProductCategory and ProductSubcategory.
- **UspTryBidProduct –** This SP will add bids to the auctioned products. It contains three parameters: @ProductID, @CustomerID and @BidAmmount. The minimum bid amount is the current highest bid price, stored as CurrentBidPrice in Auctions.AuctionProducts. The

maximum bid is also stored in the previously mentioned table. It is considered that the products are inserted before the auctions starts, therefore a bid is only valid if it happens after the beginning of the auction *(defined in configuration table)*. Regarding the upper bound of the time interval for a valid bid, depending on the product, it is consider the time limit for the bid the earlier date of the following two: expire date of the product in auction or stop bid of the overall auction. If the bid amount is not specified the registered bid results from the sum of highest current bid plus the minimum increment for a bid. In the case the bid amount exceeds the maximum bid price the bid is accepted but registered as the maximum bid value and the status updated to 4 in the auctions table as sold.

    - Tables used: Auctions.AuctionProducts, Auctions.Bids, Auctions.Config and Sales.Customer.

- **UspSearchForAuctionBasedonProductName –** This SP allows to search over the auctioned products, based on their names using a wildcard search. while returning product details. It contains three parameters: @Productname, @StartingOffSet and @ NumberOfRows where the last two are optional. Only active auction products (status=1) are displayed in the wildsearch.
    - Tables used: Auctions.AuctionProducts and Production.Product.
- **UspListBidsOffersHistory –** This SP will return an historic view of all bids for a specific customer, within a time interval. In case bit=1 only active products in auction are displayed while bid history for products no longer active in auction are also displayed if bit=0. It contains four parameters: @CustomerID, @StartTime, @EndTime and @Active. The following information is displayed in this list: product name, maximum bid allowed, date and amount of bid, status of the product in the auction and end date of the auction that can be canceled, sold or expired auction date.
    - Tables used: Sales.Customer, Auctions.AuctionProducts, Production.Product, Auctions.Bids.
- **UspRemoveProductFromAuction –** This SP will remove products from auction. Even if a product already has bids, it can always be removed. It contains one parameter, which is @ProductID. It was considered that only products that are currently active in the auction (status=1) can be removed from the auction. The status of removed products is set to 0.
    - Tables used: AuctionProducts.
- **UspUpdateProductAuctionStatus –** This SP will update the status of all products in auction. We have defined four different status for each product in auction: **1 - active**, **2 - expired**, **0 - cancelled** and **4 - sold**. Status 1, 0 and 4 are updated by the other SP manually therefore the

expired status is the only status truly being updated in this SP. The highest bid for every product is updated each time there is a bid therefore only the status needs to be updated. As explained in the SP for try bid, we consider the time limit for the bid the earlier date of the following two: expire date of the product in auction or stop bid of the overall auction therefore if the current time of the status update exceeds one of these dates then the product in the auction is expired (status=2).

- Tables used: AuctionProducts, Auctions.Config

iv. Stock clearance – Handling Errors

The following error messages were created to handle invalid arguments and inconsistencies in the usage of the different SP:

- **UspAddProductToAuction**
  - Errors:
    - 50001 – This error appears if the product that is being added to auction does not exist, is not valid, is being auctioned or was already auctioned;
- **UspTryBidProduct:**
  - Errors:
    - 50002 – In case the bid does not happen in a valid time window;
    - 50003 – In case the bid amount is below the highest current bid for said product;
    - 50004 – In case the product was not auctioned or is no longer in auction (status different than 1)*;*
    - 50005 – In case the customer does not exist in the database*;*
- **UspSearchForAuctionBasedonProductName**
  - Errors:
    - 50701 – No active auction matches the wildcard search name;
    - 50007 – Wildcard search contains less than 3 characters;
    - 50008 – In case an invalid number of rows and a starting offset is given;
- **UspListBidsOffersHistory**
  - Error: 50005 – In case the customer does not exist in the database.
- **UspRemoveProductFromAuction**

- Error: 50004 – The specified product is not in the list of active auction products (status=1).

v. Stock clearance – Views

Adventure Works leadership team also required a real time report, that allows them to evaluate the financial impact of the online auction campaign. They want to measure its level of success and understand whether it is worth repeat it, in the future. We have developed two views, which will allow them to see just that:

- **Auction.vRevenue** – This view combines information from four tables: '*Product*', '*AuctionProducts*', '*Bids*' and '*ProductSubcategory*'. It calculates '*AverageCost'* based on the standard cost by product, *'AuctionSellPrice'* based on the highest bid for the product and *'AuctionReturn'*, the percentage between the previous two values, for all products sold or expired on the auction (status=2 or status=4). Column *'NrAuctions'* was created to count the number of auctions of the product. The results are grouped by '*ProductSubcategoryID'* and '*ProductID'*. This way a user can aggregate information by subcategory. Leadership team will also be able to understand if the auctioned products are being sold, on average, at less than 70%.
- **Auction.vSales** – This view combines information from four tables: '*SalesOrderHeader'*, '*SalesOrderDetail'*, '*Product'* and '*ProductSubcategory'*. It also calculates '*AverageSales'* and '*QtySales'* of AdventureWorks' total sales of products that were properly shipped and, which are not accessories (since these are excluded from the auction campaign). The results are grouped by '*ProductSubcategoryID'* and '*ProductID'*. When querying data from both views, leadership team will have some insight over how much sales from auction represents, when compared with company's total sales.

III. **Results and Discussion** - Business problem (Brick and Mortar Stores)

There is a conflict of interests, because AdventureWorks wants to open its first two physical stores, in the United States. However, the company does not want to compete with its resellers. Based on that premise, cities where company's 30 best customers are located, will be excluded. We were asked to propose and justify, the two best cities for these new stores to be located.

In order to solve the business problem, we have queried the information from several tables: '*Store'*, '*BusinessEntity'*, '*BusinessEntityAddress'*, '*Address'*, '*StateProvince'*, '*CountryRegion'*, '*Customer'* and '*SalesOrderHeader'*. Only stores located in the US and which correspond to main offices (on the shipping type there were only duplicate values) are considered. First there was a sub selection from the top 30 stores. A selection of the top 10 cities considering average sales that were not present in the mentioned top 30 was made after. To calculate the average sales per store, we used '*TotalDue'* ('*SubTotal'* + '*TaxAmt'* + '*Freight'*) divided by the number of stores. Only sales with '*Status'* as shipped were considered. T-SQL provided in brickandmortarstores.sql.

Table 1 – Average Sales of our selected top 10 Cities

	Country	State	City	AverageSales
1	United States	California	Gilroy	481860,9122
2	United States	Georgia	Austell	475892,0433
3	United States	Texas	Mesquite	471689,9206
4	United States	Utah	Park City	470639,545
5	United States	Virginia	Chantilly	469525,749
6	United States	Minnesota	Minneapolis	460174,8591
7	United States	New Hampshire	Plaistow	452727,9318
8	United States	North Carolina	Winston-Salem	445392,9589
9	United States	Wisconsin	Racine	442256,8765
10	United States	Washington	Longview	435986,1289

IV. Conclusion

We have managed to extend AdventureWorks database model to support the new online auction campaign, by creating a new schema as well as specific stored procedures and proper errors. Two different views were created, that when combined, will allow the management team to access the pros and cons of this new campaign, as well as the true financial impact over AdventureWorks sales.

Finally, it was requested from us to suggest two cities, where AdventureWorks should open their first two brick and mortar stores, without competing directly with their 30 best costumers. After we removed those from our analysis, we chose to identify the states and cities, where the next 10 best customers are located (please refer to *Table 1*). Following that rational, we would suggest that AdventureWorks should open the first two Brick and Mortar stores in Gilroy (California) and Austell (Georgia).

T-SQL Code for Brick-and-Mortar stores

```
USE AdventureWorks; GO
SELECT Country, State, City, (SUM(TotalDue) / COUNT(Store)) as AverageSales
FROM (SELECT cr.Name AS Country, sp.Name AS State, a.City,  s.Name AS Store, SUM(SOH.TotalDue) as TotalDue
FROM Sales.Store as s
INNER JOIN Person.BusinessEntity as be ON s.BusinessEntityID = be.BusinessEntityID
INNER JOIN Person.BusinessEntityAddress as BEA ON be.BusinessEntityID = BEA.BusinessEntityID
INNER JOIN Person.Address as a ON BEA.AddressID = a.AddressID
INNER JOIN Person.StateProvince as sp ON a.StateProvinceID = sp.StateProvinceID
INNER JOIN Person.CountryRegion as cr ON sp.CountryRegionCode = cr.CountryRegionCode
INNER JOIN Sales.Customer as c ON c.StoreID = s.BusinessEntityID
LEFT OUTER JOIN Sales.SalesOrderHeader as SOH ON c.CustomerID = SOH.CustomerID
WHERE BEA.AddressTypeID = 3 AND cr.CountryRegionCode = 'US' AND SOH.Status = 5
GROUP BY cr.Name, sp.Name, a.City, s.Name) AS TopCity
WHERE City not in ( SELECT City FROM ( SELECT cr.Name AS Country, sp.Name AS State, a.City,  s.Name AS Store,
  SUM(SOH.TotalDue) as TotalDue FROM Sales.Store as s
INNER JOIN Person.BusinessEntity as be ON s.BusinessEntityID = be.BusinessEntityID
INNER JOIN Person.BusinessEntityAddress as BEA ON be.BusinessEntityID = BEA.BusinessEntityID
INNER JOIN Person.Address as a ON BEA.AddressID = a.AddressID
INNER JOIN Person.StateProvince as sp ON a.StateProvinceID = sp.StateProvinceID
INNER JOIN Person.CountryRegion as cr ON sp.CountryRegionCode = cr.CountryRegionCode
INNER JOIN Sales.Customer as c ON c.StoreID = s.BusinessEntityID
LEFT OUTER JOIN Sales.SalesOrderHeader as SOH ON c.CustomerID = SOH.CustomerID
WHERE BEA.AddressTypeID = 3 AND cr.CountryRegionCode = 'US' AND SOH.Status = 5
GROUP BY cr.Name, sp.Name, a.City, s.Name ORDER BY TotalDue DESC
OFFSET 0 ROWS FETCH NEXT 30 ROWS ONLY) top30 ) GROUP BY Country, State, City
ORDER BY AverageSales DESC OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY
GO
```