# Part 3 – extension of the database

Our approach consists in creating a VIEW that responds to business requirements described in the stock clearance part of the project, a TABLE containing data about the current bid and a TABLE containing bid history.

The VIEW will be used later to populate the auctions.

We used the following assumptions to design our model extension:

* Products can be auctioned more than once
* ProductID should be replaced by AuctionID when applicable

The scripts are the following:

DROP TABLE IF EXISTS Auction.AuctionedProductsBidHistory

CREATE TABLE Auction.AuctionedProductsBidHistory (

productid int,

productbidnumber int,

customerid int,

bid decimal(10,2),

ts datetime

primary key (productid, productbidnumber)

);

DROP TABLE IF EXISTS Auction.AuctionedProducts;

CREATE TABLE Auction.AuctionedProducts (

productid int PRIMARY KEY,

customerid int,

initialbidprice decimal(10,2),

maxbidprice decimal(10,2),

defaultbidincrease decimal(10,2),

currentbid decimal(10,2),

startdate datetime,

expiredate datetime,

lastUpdated datetime,

currentbidactive bit,

status varchar(50)

);

CREATE VIEW Auction.EligibleProducts AS

with pv as (

select productid, sum(quantity) quantity

from production.productinventory

group by productid

having sum(quantity) > 2)

select p.productId,

case

when makeflag=0 or pc.name='Components' then 0.75

else 0.25

end \* listprice initialbid,

0.02 bidincrease,

listprice maxbidlimit

from production.product p

left outer join production.ProductCategory pc

on p.productsubcategoryid = pc.productcategoryid

inner join pv

on p.productid = pv.productid

where (pc.name != 'Accessories' or pc.name is null)

and sellenddate is null

and DiscontinuedDate is null

and listprice >= 50

order by 1

Then anonymization of data was required. Many techniques can be used here:

* Masking
* Randomization
* Encrypting

We tested multiple solutions and we finally used the following randomizations for sensitive data:

Create Function [dbo].[RemoveNonAlphaCharacters](@Temp VarChar(1000))

Returns VarChar(1000)

AS

Begin

Declare @KeepValues as varchar(50)

Set @KeepValues = '%[^a-z]%'

While PatIndex(@KeepValues, @Temp) > 0

Set @Temp = Stuff(@Temp, PatIndex(@KeepValues, @Temp), 1, '')

Return @Temp

End

UPDATE [Person].[EmailAddress]

SET [EmailAddress] = SUBSTRING(CONVERT(varchar(40), NEWID()),0,9)+'@'+SUBSTRING(CONVERT(varchar(40), NEWID()),0,9)+'.com'

UPDATE [Person].[Password]

SET PasswordHash = CONVERT(varchar(40), NEWID())+CONVERT(varchar(40), NEWID())+CONVERT(varchar(40), NEWID())+SUBSTRING(CONVERT(varchar(40), NEWID()),0,8),

PasswordSalt = SUBSTRING(CONVERT(varchar(40), NEWID()),0,10)

UPDATE [Person].[Person]

SET [FirstName] = 'A'+lower(dbo.RemoveNonAlphaCharacters(SUBSTRING(CONVERT(varchar(40), NEWID()),0,15))),

MiddleName = 'A'+lower(dbo.RemoveNonAlphaCharacters(SUBSTRING(CONVERT(varchar(40), NEWID()),0,15))),

LastName = 'A'+lower(dbo.RemoveNonAlphaCharacters(SUBSTRING(CONVERT(varchar(40), NEWID()),0,15)))

UPDATE [Person].[PersonPhone]

SET [PhoneNumber] = 123+'-'+123+'-'+123

# 4 – Stock clearance

## Stored Procedures

We provide error handling to most procedures. We did multiple tests to ensure that both business rules and exceptions were acknowledged.

As we mentioned in the previous chapter, ProductID should be replaced by AuctionID to support multiple auctions per product.

The current auction status is determined by many fields:

* currentbidactive (flag)
* status
* startdate
* enddate
* lastupdated

These fields are then updated according to the business case attributed to the procedure.

The procedures are as follows:

CREATE PROCEDURE [dbo].[uspAddProductToAuction](

@ProductID int,

@ExpireDate datetime = null,

@InitialBidPrice decimal(10,2) = null

) AS

DECLARE @EligibleProducts int = 0,

@AuctionedProducts int = 0,

@AuctionID int = 1

SELECT TOP 1 @EligibleProducts=1

from Auction.EligibleProducts

WHERE productid = @ProductID

SELECT @AuctionID = max(auctionid)+1

FROM Auction.AuctionedProducts

IF @AuctionID IS NULL

BEGIN

SET @AuctionID = 1

END

BEGIN

IF @EligibleProducts = 1

BEGIN

BEGIN TRY

INSERT INTO Auction.AuctionedProducts

SELECT @AuctionID

,@ProductID

,0

,coalesce(@InitialBidPrice,initialbid)

,maxbidlimit

,bidincrease

,coalesce(@InitialBidPrice,initialbid)

,getdate()

,coalesce(@ExpireDate,DATEADD(day,7,getdate()))

,getdate()

,1

,null

from Auction.EligibleProducts

WHERE productid = @ProductID

END TRY

BEGIN CATCH

THROW 50000, 'An error has occured while trying to insert the record in the AuctionedProducts table.', 1;

END CATCH

BEGIN TRY

INSERT INTO Auction.AuctionedProductsBidHistory

SELECT auctionid

,1

,productid

,0

,currentbid

,lastUpdated

FROM Auction.AuctionedProducts

WHERE auctionid = @AuctionID

PRINT 'Product auctioned.'

END TRY

BEGIN CATCH

THROW 50000, 'An error has occured while trying to insert the record in the AuctionedProductsBidHistory table.', 1;

END CATCH

BEGIN TRY

UPDATE Production.productinventory

SET Quantity = Quantity-1

WHERE productid = @ProductID

END TRY

BEGIN CATCH

THROW 50000, 'An error has occured while updating a record in the productinventory table.', 1;

END CATCH

END

ELSE IF @EligibleProducts = 0

PRINT 'This product id is not auctionable.'

END

--EXEC [dbo].[uspAddProductToAuction] 515;

--EXEC [dbo].[uspAddProductToAuction] 515;

--EXEC [dbo].[uspAddProductToAuction] 59915;

CREATE PROCEDURE uspListBidPurchaseHistory (

@CustomerID int,

@StartTime datetime,

@EndTime datetime

) AS

select \*

from Auction.AuctionedProducts

where customerid = @CustomerID

and currentbidactive = 0

and lastupdated between @StartTime and @EndTime

and status in ('Pending Payment','Shipping','Shipped','Delivered')

CREATE PROCEDURE uspListBidsOffersHistory (

@CustomerID int,

@StartTime datetime,

@EndTime datetime,

@Active bit = 1

) AS

SELECT \*

FROM Auction.AuctionedProductsBidHistory h

INNER JOIN Auction.AuctionedProducts a on a.productid = h.productid

WHERE customerid = @CustomerID

and ts between @StartTime and @EndTime

AND currentbidactive in (1,@Active)

CREATE PROCEDURE uspRemoveProductFromAuction (

--@AuctionID int

@ProductID int

) AS

DECLARE @IsActive bit

SELECT @IsActive = currentbidactive

FROM Auction.AuctionedProducts

WHERE productid = @ProductID

--WHERE AuctionID = @AuctionID

IF @IsActive = 1

BEGIN

UPDATE Auction.AuctionedProducts

SET currentbidactive = 0,

expiredate = getdate(),

lastupdated = getdate(),

status = 'Canceled'

WHERE productid = @ProductID

--WHERE AuctionID = @AuctionID

PRINT 'Product id successfully deactivated.'

END

ELSE IF @IsActive = 0

BEGIN

PRINT 'Product id already deactivated.'

END

ELSE

BEGIN

PRINT 'Product id not found.'

END

--EXEC uspRemoveProductFromAuction 515;

--EXEC uspRemoveProductFromAuction 515;

--EXEC uspRemoveProductFromAuction 9999;

CREATE PROCEDURE uspSearchForAuctionBasedOnProductName (

@Productname varchar(50),

@StartingOffSet varchar(4) = null,

@NumberOfRows varchar(4) = null

) AS

DECLARE @query nvarchar(2000)

SET @query = N'SELECT count(1) over () TotalCount, p.\*, ap.\*

FROM [Production].[Product] p

INNER JOIN [Auction].[AuctionedProducts] ap

ON p.ProductID = ap.productid

WHERE Name LIKE ''%'+@Productname+'%''

AND currentbidactive = 1

ORDER BY Name '

IF @StartingOffSet is not null

SET @query +=N'OFFSET '+@StartingOffSet+' ROWS '

IF @NumberOfRows is not null

SET @query +=N'FETCH NEXT '+@NumberOfRows+' ROWS ONLY '

EXECUTE sp\_executesql @query

CREATE PROCEDURE uspTryBidProduct (

--@AuctionID [int],

@ProductID [int],

@CustomerID [int],

@BidAmount [int] = null

) AS

DECLARE @CustomerExists int = 0,

@ProductIsAuctioned int = 0,

@MaxBidLimit decimal(10,2) = 0,

@NextBidNumber int = 1,

@CurrentBid decimal(10,2) = null

SELECT TOP 1 @CustomerExists = 1

FROM Sales.Customer

WHERE CustomerID = @CustomerID

SELECT TOP 1 @ProductIsAuctioned = 1,

@MaxBidLimit = maxbidprice

FROM Auction.AuctionedProducts

WHERE ProductID = @ProductID

--WHERE AuctionID = @AuctionID

AND currentbidactive = 1

SELECT TOP 1 @NextBidNumber = MAX(productbidnumber)+1

FROM Auction.AuctionedProductsBidHistory

WHERE ProductID = @ProductID

--WHERE AuctionID = @AuctionID

SELECT TOP 1 @CurrentBid = currentbid

FROM Auction.AuctionedProducts

WHERE productid = @ProductID

--WHERE AuctionID = @AuctionID

BEGIN TRY

IF @BidAmount > @MaxBidLimit

BEGIN

PRINT 'Bid amount over the price limit.'

END

ELSE IF @BidAmount <= @CurrentBid

BEGIN

PRINT 'Bid amount is less than current bid.'

END

ELSE IF @CustomerExists = 0

BEGIN

PRINT 'Customer ID does not exist.'

END

ELSE IF @ProductIsAuctioned = 0

BEGIN

PRINT 'Product is not being auctioned.'

END

ELSE IF @CustomerExists = 1

AND @ProductIsAuctioned = 1

BEGIN

UPDATE Auction.AuctionedProducts

SET customerid = @CustomerID

currentbid = coalesce(@BidAmount,currentbid+defaultbidincrease),

lastUpdated = getdate()

WHERE productid = @ProductID

--WHERE AuctionID = @AuctionID

INSERT INTO Auction.AuctionedProductsBidHistory

SELECT productid,

@NextBidNumber,

@CustomerID,

currentbid,

lastUpdated

FROM Auction.AuctionedProducts

WHERE productid = @ProductID

--WHERE AuctionID = @AuctionID

PRINT 'Bid successfully placed.'

END

END TRY

BEGIN CATCH

THROW 50000,'The bid was not placed because of an error.',1

END CATCH

--EXEC uspTryBidProduct 514,1;

--EXEC uspTryBidProduct 514,1;

--EXEC uspTryBidProduct 9999,1;

--EXEC uspTryBidProduct 514,9999;

--EXEC uspTryBidProduct 514,1,999999

## Views

We developed only one view to support the analytic needs for a possible report:

CREATE VIEW Auction.AuctionSalesAnalysis AS

with s as (

select productid, sum(linetotal) linetotal

from [Sales].[SalesOrderDetail]

group by productid

)

SELECT p.name productName,

pc.name productSubCategoryName,

avg(currentbid) averageBid,

avg(StandardCost) standardCost,

sum(currentbid) totalBids,

CASE WHEN sum(currentbid) / sum(StandardCost) < 0.95 then 1 else 0 end AuctionUnder95pctOfCost,

cast(100\*sum(currentbid) / sum(StandardCost) as varchar(max)) + '%' marginVsCost,

cast(100\*sum(currentbid) / (coalesce(sum(LineTotal),0)+sum(currentbid)) as varchar(max)) + '%' propTotalSales

FROM [AdventureWorks2014].[Auction].[AuctionedProducts] ap

INNER JOIN production.Product p ON ap.productid = p.productid

LEFT OUTER JOIN production.ProductCategory pc ON p.productsubcategoryid = pc.ProductCategoryID

LEFT OUTER JOIN s on ap.productid = s.productid

where [currentbidactive] = 0

and status in ('Pending Payment','Shipping','Shipped','Delivered')

group by p.name, pc.name

# Part 5 – Brick & Mortar store

We used the following query to determine the 31st and 32nd largest customers by city:

with t as (

SELECT

a.[City]

,sum(s.TotalDue) over (partition by p.[BusinessEntityID]) TotalDue

FROM [Person].[Person] p

INNER JOIN [Person].[BusinessEntityAddress] bea

ON bea.[BusinessEntityID] = p.[BusinessEntityID]

INNER JOIN [Person].[Address] a

ON a.[AddressID] = bea.[AddressID]

INNER JOIN [Person].[StateProvince] sp

ON sp.[StateProvinceID] = a.[StateProvinceID]

INNER JOIN [Person].[CountryRegion] cr

ON cr.[CountryRegionCode] = sp.[CountryRegionCode]

INNER JOIN [Sales].[Customer] c

ON c.[PersonID] = p.[BusinessEntityID]

INNER JOIN [Sales].[SalesOrderHeader] s

ON c.CustomerID = s.CustomerID

WHERE c.StoreID IS NULL

and cr.CountryRegionCode = 'US'

)

select city, max(totaldue) ranking

from t

group by city

order by 2 desc offset 30 rows fetch next 2 rows only