Nova IMS – Information Management School

Managing Relational and Non-Relational Data

2018/2019

**SQL Project**

Joana Ricarte, nº20170983

Tiago Marques, nº20170998

## **Resumo**

Realizado no âmbito da disciplina “Managing Relational and Non-Relational Data”, este relatório pretende descrever alguns processos adotados para resolução de algumas questões de negócio propostas para uma empresa fictícia de produtos ligados a bicicletas, a *Adventure Works* - escoamento de Stock e análise das duas cidades potencialmente mais adequadas para abertura de lojas físicas.

### **Introdução**

A extração e a interpretação correta de dados recolhidos e associados a um determinado negócio, podem dar-nos respostas e sugerir estratégias relevantes para o crescimento desse mesmo negócio.

### **Adventure Works**

Este trabalho foi desenvolvido recorrendo aos dados de uma empresa fictícia de venda de produtos ligados ao ciclismo – A Adventure Works. A versão da base de dados utilizada foi a de 2014, que contém informação relacionada com as mais diferentes entidades da empresa – Recursos Humanos, Clientes, Produção, Compras e Vendas.

A Adventure Works tem um total de 290 empregados em toda a sua estrutura. Quanto à carteira de clientes, estão incluídas mais de 700 lojas às quais a empresa atua como revendedor, e mais de 19000 clientes individuais em todo o mundo, aos quais a empresa vende diretamente.

A atividade da empresa está espalhada por todo o mundo, estando presente em países como os Estados Unidos da América, Canadá, Austrália, Reino unido, Alemanha, França, etc. O modelo de negócio da Adventure Works divide os clientes em duas categorias – lojas de retalho ou clientes individuais.

Importante salientar que as vendas a clientes individuais são feitos exclusivamente online e que a empresa não possuí lojas físicas.

**Escoamento de Stock**

Perante um problema relatado da empresa, relacionado com a capacidade de diminuição de stock antes do lançamento de novos modelos, foi sugerida uma nova abordagem de leiloamento de alguns produtos.

Os produtos, cujos novos modelos fossem lançados em menos de duas semanas, deveriam estar disponíveis para leilão, no site online, com valores de licitação iniciais entre 25-75% do valor de venda.

Para este procedimento, será necessário estender a base de dados para suportar esta nova funcionalidade do website da companhia.

Esta campanha estará prevista ser sazonal, e nas últimas duas semanas de novembro, sendo esperada uma maior afluência de licitadores e um maior volume de licitações aquando a Black Friday (a acontecer neste período).

**Procedimento**

Extensão da Base de Dados

A abordagem utilizada para responder a esta necessidade de negócio foi de criar uma view, uma tabela que contém os dados da licitação mais recente e outra tabela que contém o histórico das licitações. A view deverá ser utilizada com o objetivo de popular os dados dos leilões.

Foram assumidos alguns pressupostos para a extensão da base de dados proposta:

* Os produtos podem ser leiloados mais do que uma vez
* O inventário de stock de cada produto tem de ser atualizado a cada venda efectivamente realizada
* O ID do produto (ProductID) deverá ser substituído pelo ID do Leilão (AuctionID), quando aplicável.

Os scripts utilizados foram os seguintes:

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

Anonimização dos Dados

O tratamento de dados sensíveis, também pedido para ser tido em conta na elaboração desta solução, pode ser feito recorrendo a várias técnicas:

* Masking
* Randomization
* Encription (fácil reversão pelo uso de uma chave de desencriptação)

Foram testadas várias soluções, no entanto, optou-se por realizar a anonimização dos dados usando os scripts de Randomization abaixo:

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

Escoamento de Stock (Stored Procedures)

A maioria das Stored Procedures têm error handling, que foram testados a par com algumas das regras de negócio definidas.

O estado presente do leilão de um produto é determinado por vários campos:

* Flag CURRENTBIDACTIVE
* STATUS
* STARTDATE
* ENDDATE
* LASTUPDATED

Estes campos são então atualizados de acordo com o caso de negócio atribuído à stored procedure.

Abaixo encomtram-se as stored procedures desenvolvidas. No fim das stored procedures podem ser encontrados valores de teste.

* uspAddProductToAuction

Permite adicionar um produto a leilão.

Esta é talvez a procedure mais importante e verifica se o produto é elegível para leilão, e caso afirmativo atualiza a tabela de inventário de produto.

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;

* uspListBidPurchaseHistory

Devolve todos os produtos que um cliente comprou por leilão, num determinado período, e apresenta o estado da encomenda.

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')

* uspListBidsOffersHistory

Devolve o histórico de licitações de um produto para um determinado intervalo. A flag Active deverá controlar se os produtos devolvidos são só produtos leiloados ou também produtos comprados.

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)

* uspRemoveProductFromAuction

Retira o produto de um leilão, mesmo que hava licitações sobre o mesmo.

Utiliza a flag *@IsActive* para identificar se o produto se encontra ativo e aplica lógica de negócio em consequência.

Apenas a tabela principal de licitação é afetada.

CREATE PROCEDURE uspRemoveProductFromAuction (

--@AuctionID int

@ProductID int

) AS

DECLARE @IsActive bit

SELECT @IsActive = currentbidactive

FROM Auction.AuctionedProducts

WHERE productid = @ProductID

--WHRE 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;

* uspSearchForAuctionBasedOnProductName

Permite procurar um produto licitado na base de dados, com opção de paginamento controlada pelos campos *Starting Offset* e *Number of Rows*.

A procedure executa, assim, uma query de SQL dinâmica.

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

* UspTryBidProduct

Adiciona licitação de um cliente a um produto.

As tabelas de licitação (atual e histórico) são modificadas.

Antes de adicionar uma licitação, é verificado se o cliente existe, se o produto está a ser licitado, e se o limite máximo de licitação foi atingido.

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

View

Foi criada apenas uma view para suportar eventuais necessidades de análise dos dados, para um possível relatório.

Esta view contém diversas métricas agregadas por produto e subcategoria de produto:

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

**Critério para Instalação Lojas Físicas**

Ao ser planeado abrir duas lojas físicas, para venda de artigos diretamente a clientes individuais, a empresa decidiu não querer concorrer com os clientes a quem faz revenda, de forma a não prejudicar as suas vendas.

Por esse motivo, serão excluídas as cidades dos 30 melhores clientes a quem a organização faz revenda.

Procedimento

Por indicação do negócio, a escolha da cidade não deve incluir os 30 melhores clientes.

Assim, não foi tido em consideração qualquer tipo de filtro temporal nem regional.

Para recomendação da localização das duas lojas físicas da empresa, foi assim determinado que seriam escolhidas as cidades dos 31º e 32º melhores clientes da empresa, através da seguinte query:

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

### **Crítica e Conclusões**

Com este modelo de extensão, torna-se possível à empresa criar um modelo simples de leilões, mantendo os dados de inventário da empresa atualizados e ajudando no *roll out* de produtos obsoletos.

Abordagens mais complexas poderiam ser utilizadas para afinar o modelo às necessidades de negócio pretendidas.

Uma análise funcional mais detalhada, a inclusão de mais pormenores sobre o funcionamento da aplicação de leilões, e a inclusão de mais variáveis na análise de criação das lojas físicas ajudariam a tornar esta extensão de modelo relacional mais flexível e as recomendações mais fiáveis.

### **Bibliografia**

* Aulas de Managing Relational and Non-Relational Data, EDSA, 2018, NOVA IMS
* <https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure>
* <https://support.office.com/en-us/article/about-adventure-works-and-the-sample-business-database-00a88101-ef11-4a8d-8904-b9747f53c961>
* <https://docs.microsoft.com/en-us/sql/t-sql/tutorial-writing-transact-sql-statements?view=sql-server-2017>