Tips and Hints: Using the View Designer

The **View Designer** is a visual tool that can help you build views.

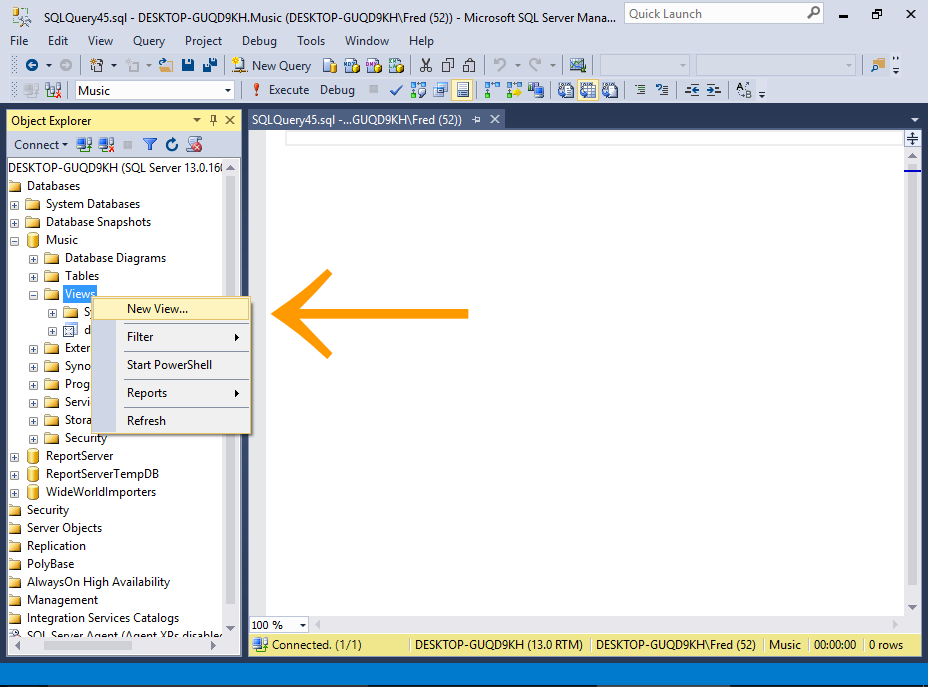
However I strongly recommend utilisng Views across related Tables and comes with more features than the Query Designer. Unlike the Query Designer the View Designer can executes the TSQL within its environment.

The View Designer is just like the Query Designer, in that it provides a visual way of designing a query. It saves you the trouble of designing the query in Query Designer, then doing the extra coding to convert the query into a view as in the previous Quick Guide – Create a View.

Note in section 8 we also cover the code approach to SQL Views.

Now we will use the View Designer to create another view.

# Launch the View Designer



Right-click on the Views node and select New View....

# Add the Tables

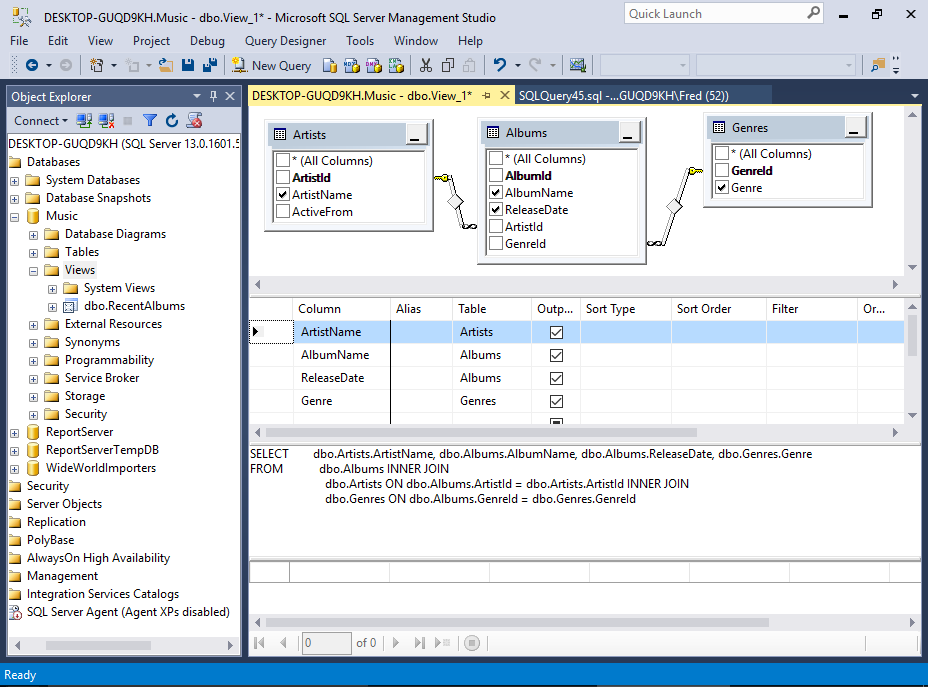
Graphical user interface, application

Description automatically generated

Select the tables that you want to include in your view.

In our case, select all and click Add, then click Close to close the dialog box.

# Design the View



You will now see the selected tables, and their relationships — just like in the Query Designer.

One difference is that the View Designer is sitting inside a query window. When we used the Query Designer, it was opened in a pop-up dialog, which prevented us from accessing any of the toolbar options, etc.

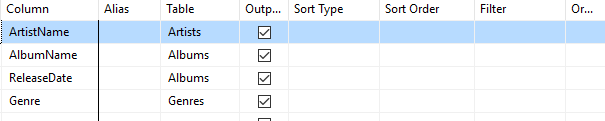
But having the View Designer opened inside a query window allows us to access the toolbar and other options as required.

Another difference is that we now have a fourth pane — the Results Pane. This allows us to execute the SQL to see what effect it has on the results before we save the view.

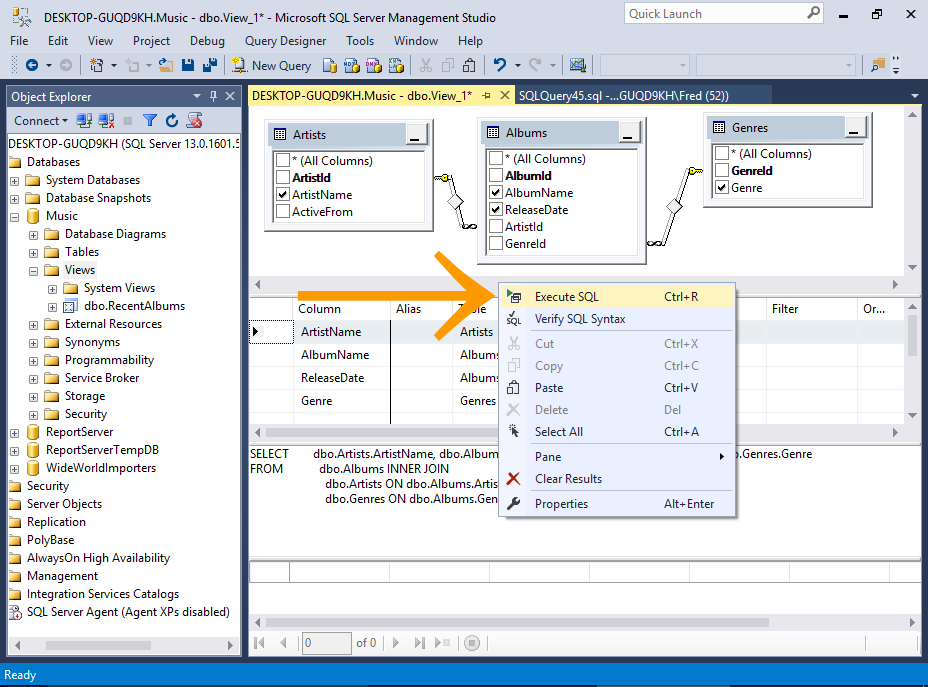
**Our Example**

We'll keep it simple this time and select four columns from the three tables. We won't provide any sort or filter criteria but you should look to also utilise these options.

Here's a close-up of the Criteria Pane:



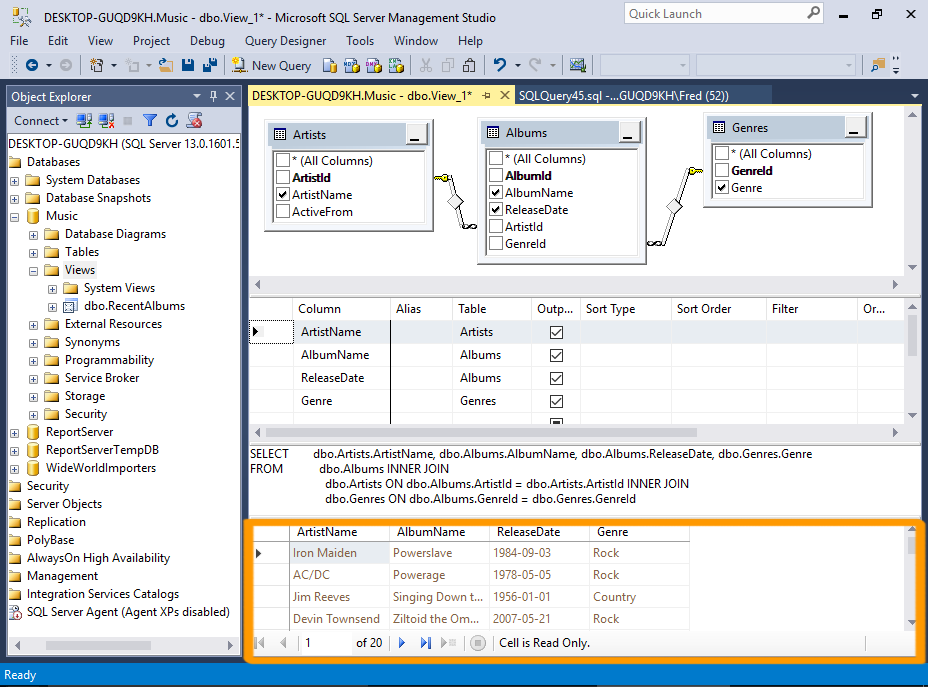
# Execute the SQL



You can test the view before you save it by executing the SQL while in the View Designer.

To do this, right-click anywhere in the design area and select Execute SQL (or press Ctrl+R on your keyboard).

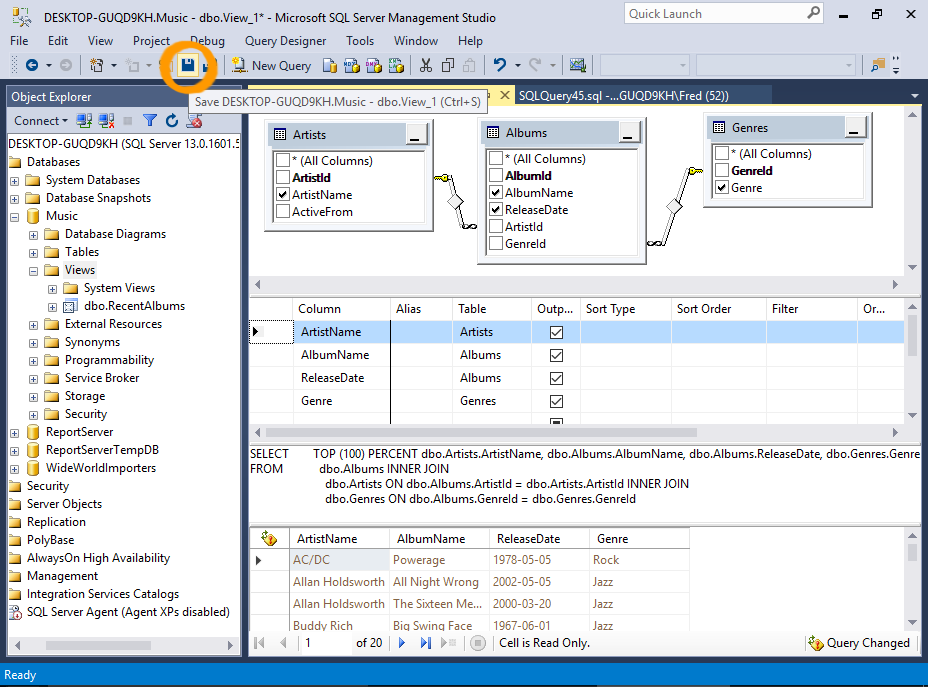
# The Results

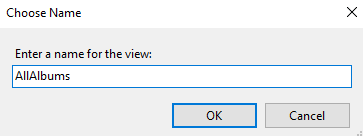


Do utilise the Sort & Filter options

The results appear in the Results Pane at the bottom.

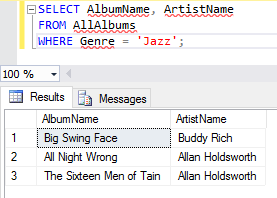
# Save the View





To save the view, click the Save icon on the toolbar, then name the view at the prompt.

# Query the View



# Code approach to Views in SQL Server

This SQL View tutorial discusses one of the basic but greatly misunderstood objects in SQL server – “Views”. Views could be looked as an additional layer on the table which enables us to protect intricate or sensitive data based upon our needs. It’s like a window exposed to the flocks and they can see very selective items inside the room through the window.

Since views are an additional layer sure they do add an overhead but there is a tradeoff for situations when they are of great help.

As we proceed, we will try to learn more about them not by theoretical explanation but by some practical examples.

Uses

We begin with creating 3 tables PRODUCTS, Customer & BOOKING. These are fictitious tables for our demo. The PRODUCTS stores data for a retail shop with a flag column IsSalable based on whose value we treat the products as Salable.

CREATE TABLE PRODUCTS

(ProductID INT PRIMARY KEY CLUSTERED,

ProductDesc VARCHAR(50) NOT NULL,

ManufacturingDate DATETIME,

ExpiryDate DATETIME,

IsSalable BIT,*--1 Salable/Active FOR 0 For NonSalable/Passive Product*

Price MONEY NOT NULL

)

Next, we have a Customer table which stores UserID and Password details for customers.

CREATE TABLE Customer

(CustID INT IDENTITY(1002,2)PRIMARY KEY CLUSTERED,

FName VARCHAR(50) NOT NULL,

LNme VARCHAR(50) NOT NULL,

UserID VARCHAR(100) NOT NULL,

Pswd NVARCHAR(100) NOT NULL DEFAULT 'password123'

)

Lastly, I have created a BOOKING table which houses all the bookings from different customers.

CREATE TABLE BOOKING

( BookingID INT IDENTITY(10,2) PRIMARY KEY CLUSTERED,

ProductID INT REFERENCES dbo.Products(ProductID),

CustID INT REFERENCES dbo.Customer(CustID),

DateOfBooking DATETIME NOT NULL,

QTY INT

)

Next, insert a few records into these tables:

INSERT INTO PRODUCTS VALUES

(1,'Biscuits','2011-09-01 00:00:00.000','2012-09-01 00:00:00.000',1,20),

(2,'Butter','2010-09-01 00:00:00.000','2011-09-01 00:00:00.000',1,30),

(3,'Milk','2011-10-01 00:00:00.000','2011-11-01 00:00:00.000',1,46)

INSERT INTO Customer (FName,LNme,UserID,Pswd) VALUES

('Alex','Groves','A.Groves@abc.com','S123'),

('Vish','Thako','V.Thako@xyz.com','G311'),

('Mansh','Mongo','M.Mongo@mno.com','M222')

INSERT INTO BOOKING (ProductID,CustID,DateOfBooking,QTY) VALUES

(1,1002,'2011-11-01 00:00:00.000',3),

(2,1004,GETDATE(),4),

(3,1006,'2011-10-01 00:00:00.000',2)

Our tables contents look like this. I know the tables are not completely normalized, for now please ignore them, these are simple demo tables.

**SELECT \* FROM Customer**

CustID FName LNme UserID Pswd

--------- -------- ---------- -------------- ---------

1002 Alex Groves A.Groves@abc.com S123

1004 Vish Thako V.Thako@xyz.com G311

1006 Mansh Mongo M.Mongo@mno.com M222

(3 row(s) affected)

**Select \* from PRODUCTS**

ProductID ProductDesc ManufacturingDate ExpiryDate IsSalable Price

---------- ------------ ----------------------- ----------------------- --------- -------

1 Biscuits 2011-09-01 00:00:00.000 2012-09-01 00:00:00.000 1 20.00

2 Butter 2010-09-01 00:00:00.000 2011-09-01 00:00:00.000 1 30.00

3 Milk 2011-10-01 00:00:00.000 2011-11-01 00:00:00.000 1 46.00

(3 row(s) affected)

**Select \* from BOOKING**

BookingID ProductID CustID DateOfBooking QTY

----------- ----------- ----------- ----------------------- -----------

10 1 1002 2011-11-01 00:00:00.000 3

12 2 1004 2011-10-09 17:31:31.790 4

14 3 1006 2011-10-01 00:00:00.000 2

(3 row(s) affected)

A customer purchases/books a product and the same gets recorded into the BOOKING table now to generate the Invoice on his name we can uses a VIEW which would help us do away with a physical table. Instead it would enable us to generate the Invoice based on the information from these 3 tables itself. Let’s see how it’s possible.

CREATE VIEW Invoice\_V

AS

SELECT C.FName

,C.LNme

,P.ProductDesc

,B.DateOfBooking

,P.Price

,B.QTY

,(B.QTY\*P.Price) AS TotalAmountPayable

FROM BOOKING B

INNER JOIN PRODUCTS P

ON B.ProductID=P.ProductID

INNER JOIN Customer C

ON B.CustID=C.CustID;

Select \* from Invoice\_V

FName LNme ProductDesc DateOfBooking Price QTY TotalAmountPayable

-------------------------------------------------- ------------------------------------

Alex Groves Biscuits 2011-11-01 00:00:00.000 20.00 3 60.00

Vish Thako Butter 2011-10-09 17:31:31.790 30.00 4 120.00

Mansh Mongo Milk 2011-10-01 00:00:00.000 46.00 2 92.00

(3 row(s) affected)

We have been able to generate the Invoice based on the 3 tables hence we have not only optimized the Invoice generation also we have saved ourselves from hosting a physical table in the database with this information.

* This is the most credible use of a VIEW; it can not only reduce apparent complexity but also prevent redundant hosting of data in the DB.

Next say there is some API which enables the Customer care executives to view the customerinformation details. Now exposing the Password might be risky, it’s strictly confidential info.

We create a View which can be exposed to the API:

CREATE VIEW dbo.CustomerInfo\_V

AS

Select CustID

,FNAME AS [FIRST NAME]

,LNME AS [LAST NAME]

,UserID

FROM dbo.Customer

We have a created a View which can be used by the API to fetch customer details –(Minus) the Password Column.

* Views can be used to prevent sensitive information from being selected, while still allowing other important data.

Views do not have a physical existence, but still they do return a set of record set as a table does, the differences is it is simply an additional layer which calls the underlying code which finally returns the record set.

When I execute the code...

Select \* from CustomerInfo\_V

...I get recordsets as I would get in a table with the only difference that the data returned is as per the below query:

Select CustID

,FNAME AS [FIRST NAME]

,LNME AS [LAST NAME]

,UserID

FROM dbo.Customer

But arguably, we still get a set of records, isn’t it? So say if there are 1 million customers in my database, wouldn’t it be cool if I have clustered/non clustered index on my view for optimized queries. But is it possible as the view doesn’t host data physically, the answer is yes. It is possible to have indexes on views. But before we find ourselves capable of creating index on views, we have to SCHEMABIND our VIEWs.

**What is SCHEMABINDING a VIEW**

Schema binding binds your views to the dependent physical columns of the accessed tables specified in the contents of the view, i.e. if CustomerInfo\_V is schema bind no one will be able to alter the dbo.Customer table unless they drop the table.

**Why would we need that?**

The answer is, it prevents your views from being orphaned. Just think that someone drops/alters the table dbo.Customer without paying any heed to our view. Now that would leave our view nowhere. Hence schema bind it, this will prevent any such accidents from happening.

Also to be able to create an index on the view you need it essentially schema bound.

Let’s make the change:

ALTER VIEW Invoice\_V

WITH SCHEMABINDING

AS

SELECT C.FName

,C.LNme

,P.ProductDesc

,B.DateOfBooking

,P.Price

,B.QTY

,(B.QTY\*P.Price) AS TotalAmountPayable

FROM dbo.BOOKING B

INNER JOIN dbo.PRODUCTS P

ON B.ProductID=P.ProductID

INNER JOIN dbo.Customer C

ON B.CustID=C.CustID;

/\* Now we are licensed to have an Index on this dbo.Invoice\_V view. \*/

CREATE UNIQUE CLUSTERED INDEX Invoice\_View\_Indx

ON dbo.Invoice\_V(Fname,LNme);

Now we have an index on the view, remember you need to have a UNIQUE CLUSTERED INDEX to be able to create a NONCLUSTERED INDEX. By which I mean I can create the below index mandatorily post the creation of the Invoice\_View\_Indx.

CREATE NONCLUSTERED INDEX Invoice\_View\_Indx2

ON dbo.Invoice\_V(ProductDesc);

* So next use of the View is to be able to create an additional index upon the db to speed up your query performance.

**Features**

Are views only meant for reading data in a customized mode? Not really views also facilitate DML (Insert/Update/Delete). But there is a set of rules which needs to be adhered to enable DMLs.

* + If you are using a view to insert data, then your view should have a single select and also all the mandatory columns of the “being edited” table must be included in the view unless the table has a default values for all NOT NULL columns of the table.
  + Secondly don’t forget, for views with “WITH CHECK” options enabled, it’s important to keep in mind that the data begin inserted qualifies in the WHERE clause of the view and is certain to be selected by the view. Simply put the data you insert is picked up while you select from your view.
  + If the view is having joins with more than one table, then most cases chances of modifying capabilities are negligible unless INSTEAD OF Triggers are in place to handle the request.

Keeping these in mind, let’s turn to an example and perform INSERTs / Updates / Deletes.

I am altering the below view as:

ALTER VIEW dbo.CustomerInfo\_V

WITH SCHEMABINDING

AS

Select CustID

,FNAME AS [FIRST NAME]

,LNME AS [LAST NAME]

,UserID

FROM dbo.Customer

WITH CHECK OPTION

* + **Insert**

INSERT INTO CustomerInfo\_V

([FIRST NAME],[LAST NAME],UserID)

VALUES ('Tyrone','Davison','T.Dav@qrs.com')

The insert happened because though the columns CustID and Pswd are mandatory but CustID is IDENTITY and PSWD has a DEFAULT. All the other mandatory data was supplied in the insert query.

SELECT \* FROM Customer

CustID FName LNme UserID Pswd

*--------- -------- ---------- --------------- ---------*

1002 Alex Groves A.Groves@abc.com S123

1004 Vish Thako V.Thako@xyz.com G311

1006 Mansh Mongo M.Mongo@mno.com M222

1008 Tyrone Davison T.Dav@qrs.com password123

(4 row(s) affected)

* + **Update**

UPDATE CustomerInfo\_V

SET [FIRST NAME]='Tyroneoorthy'

WHERE [FIRST NAME]='Tyrone'

SELECT \* FROM Customer

CustID FName LNme UserID Pswd

--------- -------- ---------- --------------- ---------

1002 Alex Groves A.Groves@abc.com S123

1004 Vish Thako V.Thako@xyz.com G311

1006 Mansh Mongo M.Mongo@mno.com M222

1008 Tyrone Davison T.Dav@qrs.com password123

(4 row(s) affected)

* + **Delete**

DELETE FROM CustomerInfo\_V

WHERE [FIRST NAME]='Tyroneoorthy'

SELECT \* FROM Customer

CustID FName LNme UserID Pswd

--------- -------- ---------- --------------- ---------

1002 Alex Groves A.Groves@abc.com S123

1004 Vish Thako V.Thako@xyz.com G311

1006 Mansh Mongo M.Mongo@mno.com M222

Displaying the View Contents

For retrieving what is under the hood of the view use,

EXECUTE SP\_HELPTEXT 'dbo.CustomerInfo\_V'

Alternatively in SSMS.

Tools > Options > Environment >Keyboard > Ctrl-F1 == SP\_HELPTEXT

From next time, to see the contents of a VIEW/StoreProcedure, simply select it and hit Ctrl+F1?

Refreshing Views

Just in case we are working with a non-schema bound view and there is some change in the underlying table, to prevent the view from producing unexpected results, we have an option to refresh the view with:

EXECUTE SP\_REFRESHVIEW 'dbo.INVOICE\_V'

This updates the metadata of a non-schema bound view.

Encrypting your Views

The “WITH ENCRYPTION” option encrypts the views by which I mean it will not be visible via SP\_HELPTEXT so in case of strict requirements where the contents of the view don’t need to be exposed this option freezes the view. It’s important to save the contents script in some archive to be able to retrieve the code for any change.

ALTER VIEW Invoice\_V

WITH ENCRYPTION

AS

SELECT C.FName

,C.LNme

,P.ProductDesc

,B.DateOfBooking

,P.Price

,B.QTY

,(B.QTY\*P.Price) AS TotalAmountPayable

FROM dbo.BOOKING B

INNER JOIN dbo.PRODUCTS P

ON B.ProductID=P.ProductID

INNER JOIN dbo.Customer C

ON B.CustID=C.CustID;

So we have encrypted the view, now if we try to EXECUTE SP\_HELPTEXT 'dbo.Invoice\_V'.

The result would be:

The text for object ' Invoice\_V ' is encrypted.

It is not advised to encrypt your views unless you have a very strong reason behind it. That was some key elements involved. I hope the next time you work with views, this tutorial will help you make better decisions.