Views Lab Manual + Northwind



Session: 2022 – 2026

**Submitted by:**

Saad Ahmad Malik 2022-CS-1

**Supervised by:**

Mr. Nazeef ul Haq

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Views**

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

**Types of views in SQL Server:**

There are the following two types of views:

1. User-Defined Views
2. System-Defined Views

First, we discuss User-Defined Views.

*I will use Database Northwind to illustrate the concept of Views in DBMS.*

**Create SQL VIEW in SQL Server**

1. CREATE VIEW view\_name AS
2. SELECT columns
3. FROM tables
4. WHERE conditions; Let us create some views.

**Method 1:** We can select all columns of a table. The following example demonstrates that:

Create View Products\_View AS

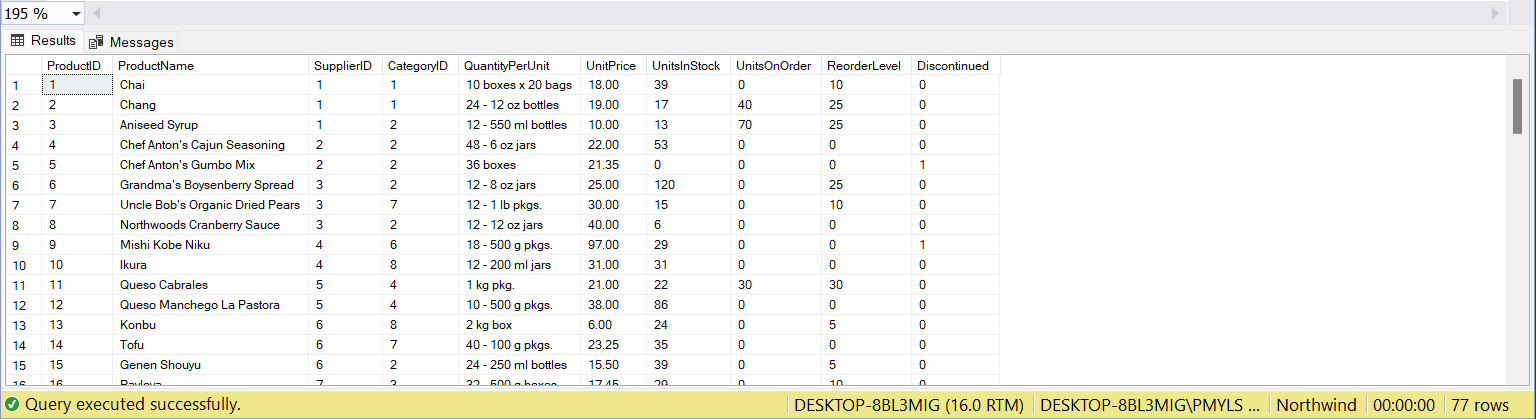
SELECT \* FROM Products

GO

**To Execute this View:**

SELECT \* FROM Products\_View

**Output:**



**Method 2:** We can select specific columns of a table. The following example demonstrates that:

Create View Products\_View\_1 AS

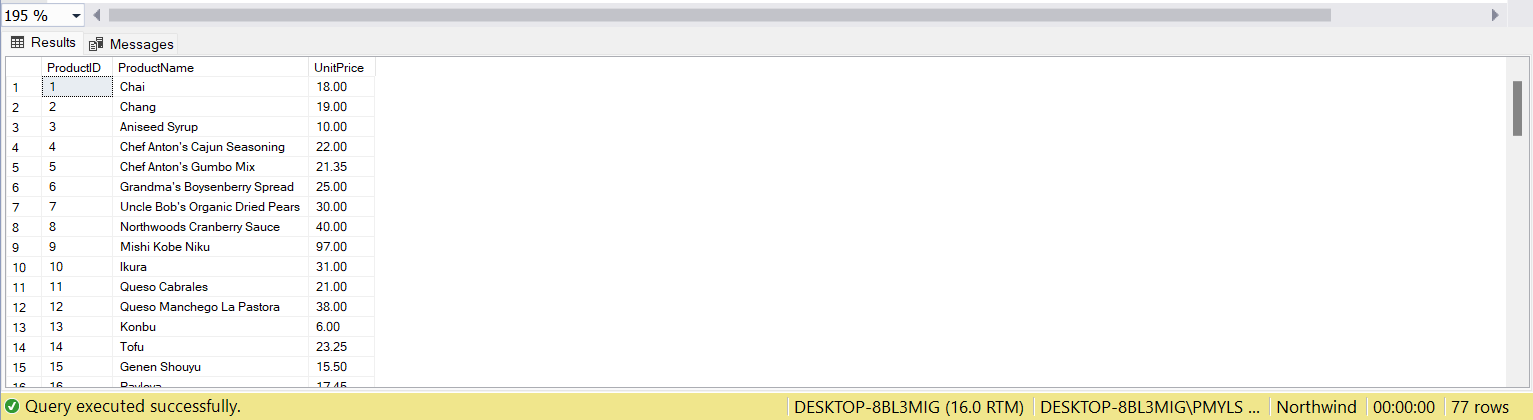
SELECT ProductID,ProductName,UnitPrice FROM Products

GO

**To Execute this View:**

SELECT \* FROM Products\_View\_1

**Output:**



**Method 3:** We can select columns from a table with specific conditions. The following example demonstrates that:

Create View Products\_View\_2 AS

SELECT ProductID,ProductName,UnitPrice

FROM Products

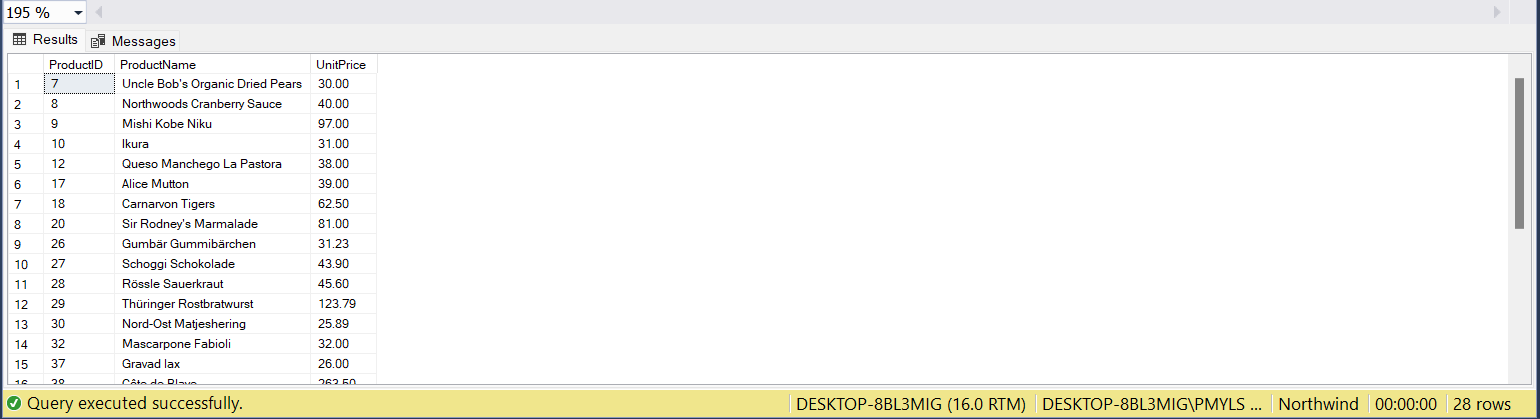
WHERE UnitPrice > 25

GO

**To Execute this View:**

SELECT \* FROM Products\_View\_2

**Output:**

****

**Method 4:** We can create a view that will hold the columns of different tables. The following example demonstrates that:

Create View Products\_View\_3 AS

SELECT ProductID,CategoryName,ProductName,UnitPrice

FROM Products

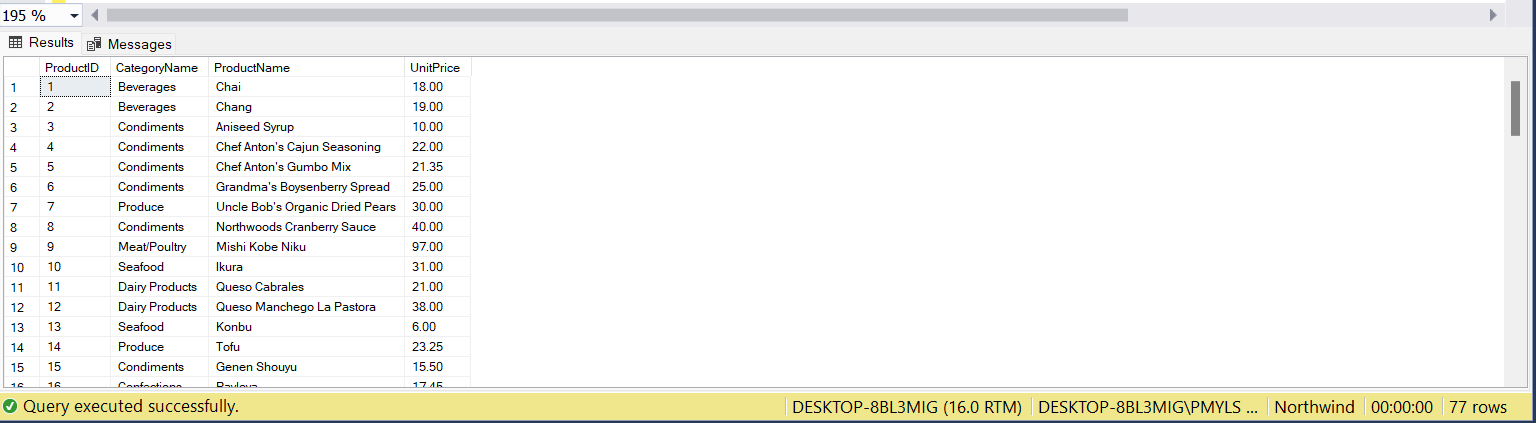
JOIN Categories ON Categories.CategoryID = Products.CategoryID

GO

**To Execute this View:**

SELECT \* FROM Products\_View\_3

**Output:**



**Retrieve Data from View in SQL** **Server**

This SQL CREATE VIEW example would create a virtual table based on the result set of the select statement. Now we can retrieve data from a view as follows:

1. Select \* from Products\_View
2. Select ProductId, ProductName, UnitPrice from Products\_View

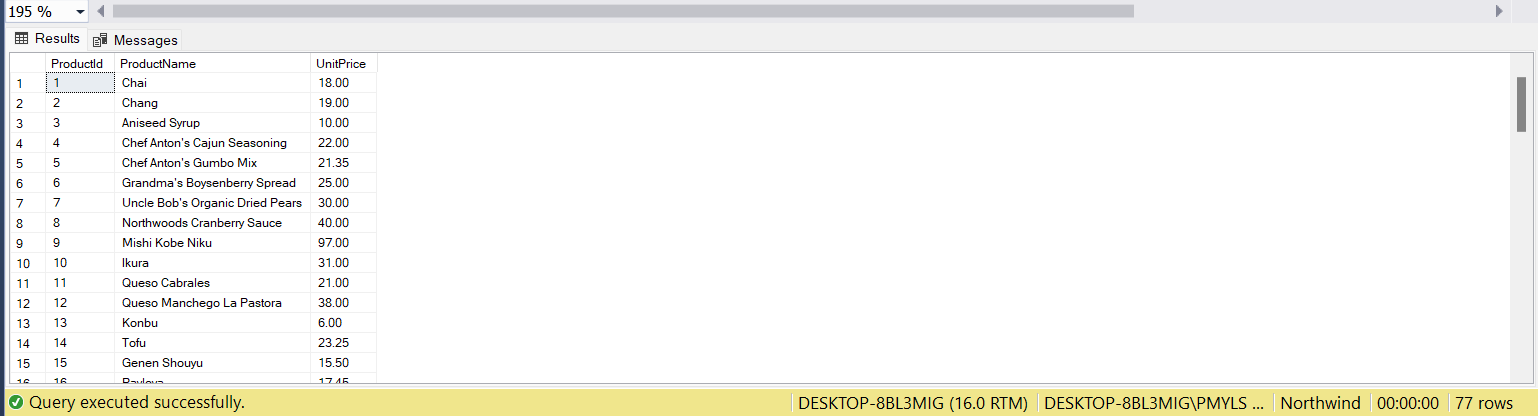


Figure: Example of selecting specific columns from a View

The preceding query shows that we can select all the columns or some specific columns from a view.

**Dropping a View in SQL Server**

We can use the Drop command to drop a view. For example, to drop the view Products\_View\_3, we can use the following statement.

1. Drop View Products\_View\_3

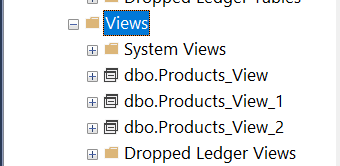


Figure: Drop View Products\_View\_3, after execution

For the moment we will create the view back.

**Renaming the View in SQL Server**

We can use the sp\_rename system procedure to rename a view. The syntax of the sp\_rename command is given below:

Sp\_Rename OldViewName , NewViewName

**Example:**

We will rename the Products\_View

Sp\_Rename Products\_View, View\_Products

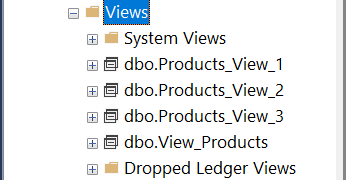


Figure: Products\_View after renaming

**Getting Information about a view:**

We can retrieve all the information of a view using the Sp\_Helptext system Stored Procedure. Let us see an example.

Sp\_Helptext View\_Products

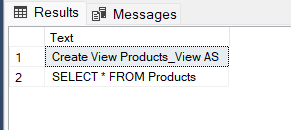


Figure: Output of the example.

**Alter View in SQL Server**

We can alter the schema or structure of a view. In other words, we can add or remove some columns or change some conditions that are applied in a predefined view. Let us see an example.

Alter View Products\_View\_3 AS

SELECT ProductID, CompanyName, ProductName, UnitPrice

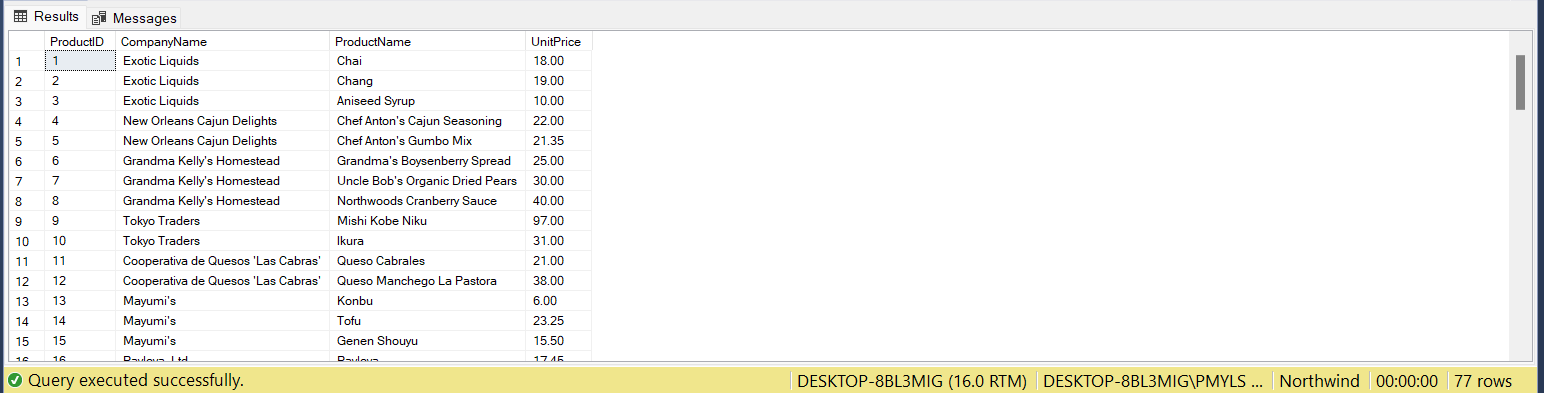
FROM Products

JOIN Suppliers ON Suppliers.SupplierID = Products.SupplierID

GO

**Output:**

SELECT \* FROM Products\_View\_3



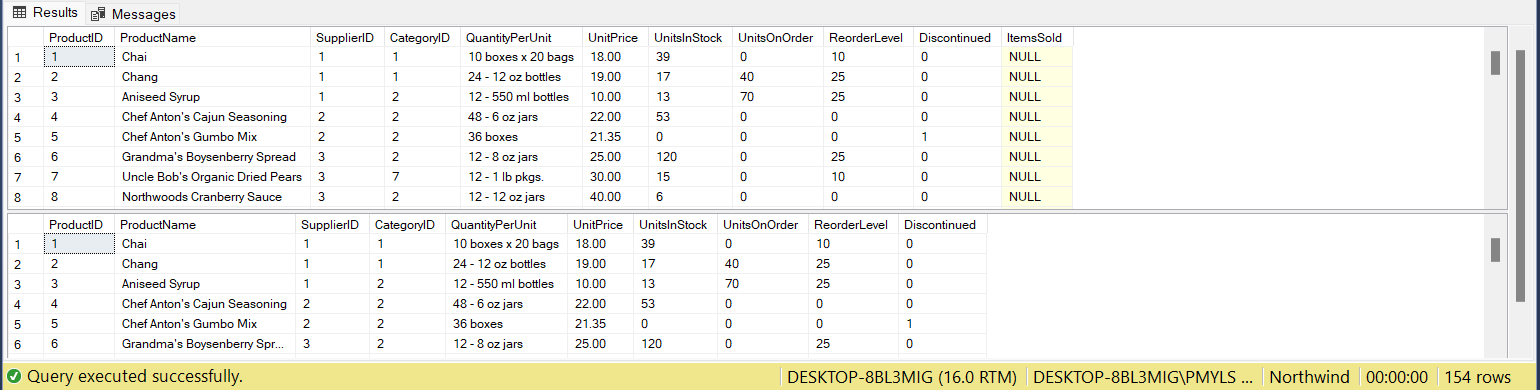
**Refreshing a View in SQL Server:**

Let us consider the scenario now by adding a new column to the table Products and examine the effect.

Alter Table Products Add ItemsSold nvarchar(50)

Select \* from Products

Select \* from View\_Products



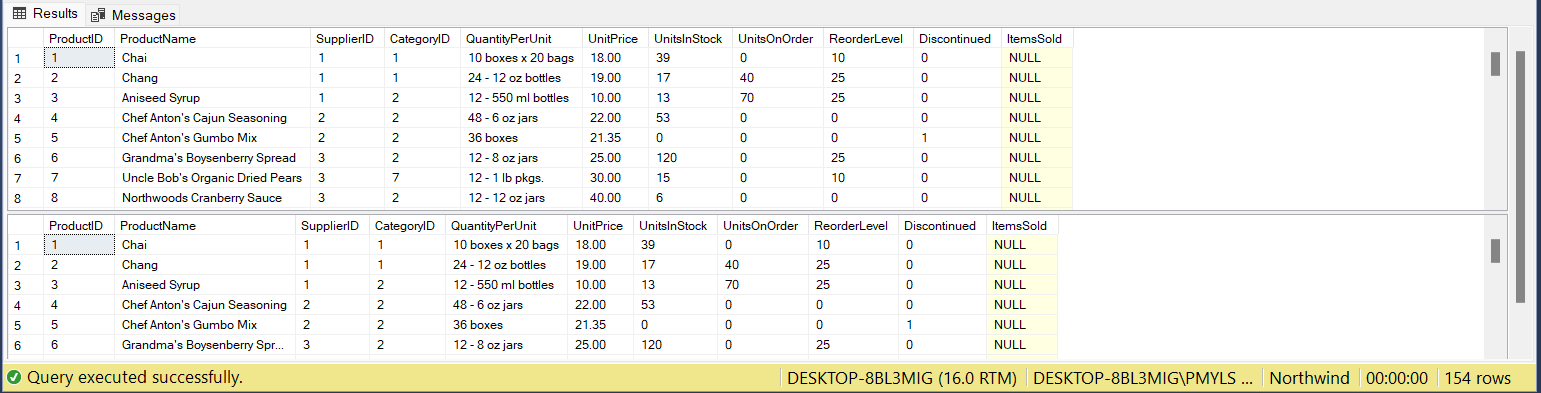
We don't get the results we expected because the schema of the view is already defined. So when we add a new column into the table it will not change the schema of the view and the view will contain the previous schema. For removing this problem, we use the system-defined Stored Procedure sp\_refreshview.

**sp\_refreshview** is a system-level Stored Procedure that refreshes the metadata of any view once you edit the schema of the table. Let's execute the following:

Exec sp\_Refreshview View\_Products

Select \* from Products

Select \* from View\_Products



**Schema Binding a VIEW**

In the previous example, we saw that if we add a new column into the table then we must refresh the view.

Such a way if we change the data type of any column in a table then we should refresh the view. If we want to prevent any type of change in a base table then we can use the concept of SCHEMABINDING. It will lock the tables being referred to by the view and restrict all kinds of changes that may change the table schema (no Alter command).

Create View Products\_View\_4

WITH SCHEMABINDING

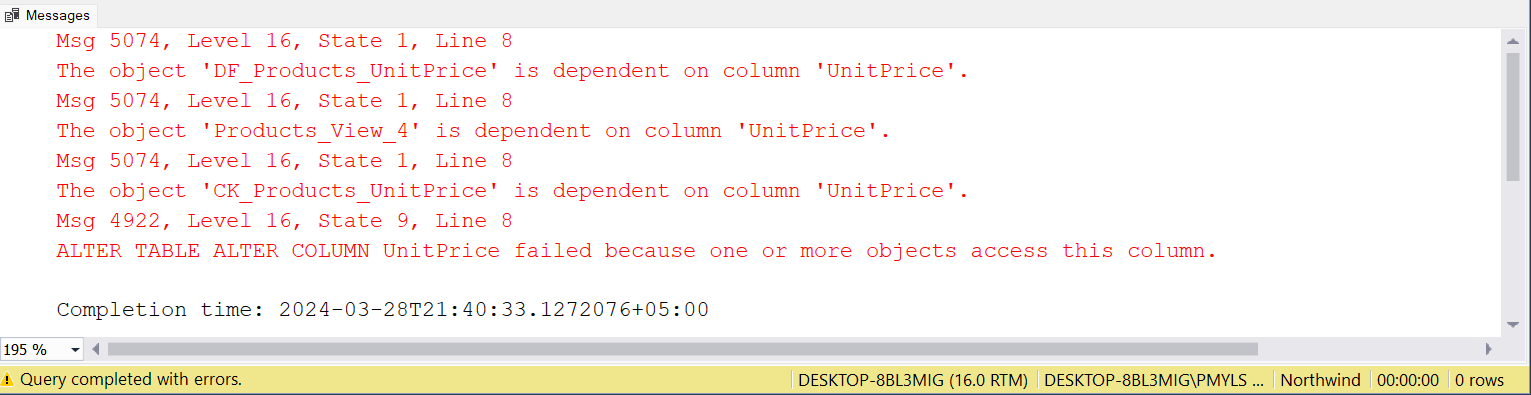
AS

SELECT ProductID,ProductName,UnitPrice

FROM dbo.Products

GO

In the preceding example, we create a view using Schema binding. Now we try to change the datatype of UnitPrice from money to int in the Base Table.

****

We find that we cannot change the data type because we used the SCHEMABINDING that prevents any type of change in the base table.

**Encrypt a view in SQL Server:**

The “WITH ENCRYPTION” option can encrypt any views. That means it will not be visible via SP\_HELPTEXT. This option encrypts the definition. This option encrypts the definition of the view. Users will not be able to see the definition of the view after it is created. This is the main advantage of the view where we can make it secure.

Create View Products\_View\_5

WITH Encryption

AS

SELECT ProductID,ProductName,UnitPrice

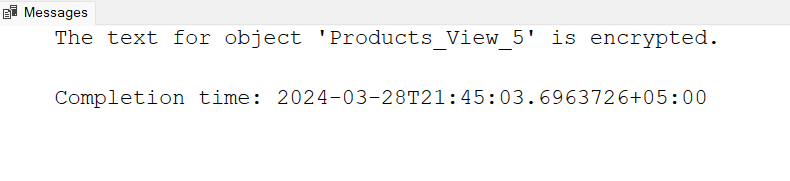
FROM Products

GO

Now we try to retrieve the definition of the view.

sp\_Helptext Products\_View\_5

**Output:**

****

**Check Option:**

The use of the Check Option in a view is to ensure that all the Update and Insert commands must satisfy the condition in the view definition.

Let us see with an example.

Create View Products\_View\_6 AS

SELECT ProductID,ProductName,UnitPrice

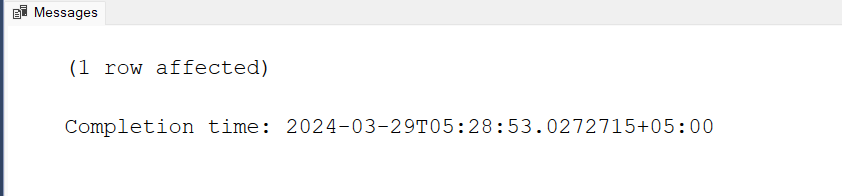
FROM Products

WHERE UnitPrice < 250

GO

In the preceding example, we create a view that contains all the data for which UnitPrice < 250 but we can insert the data for a product having Unit price more than 250 as follows.

Insert Into Products\_View\_6 values ('Caramel',270)



Now we drop the View and create it using Check option to prevent this issue as:

Create View Products\_View\_6 AS

SELECT ProductID,ProductName,UnitPrice

FROM Products

WHERE UnitPrice < 250

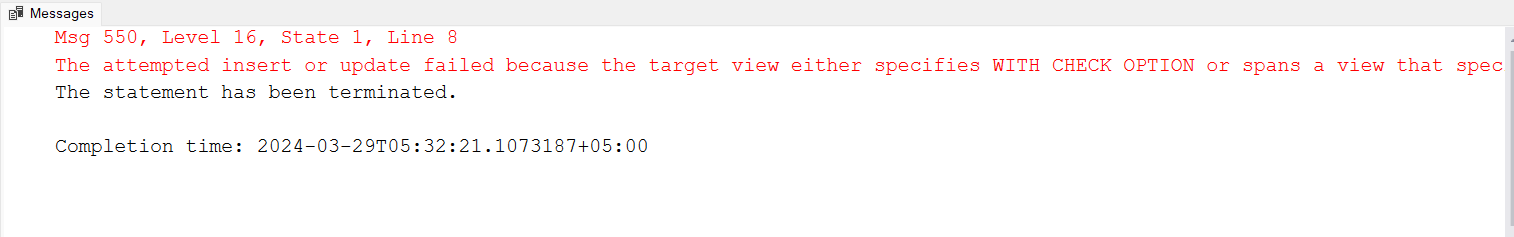
WITH Check Option

GO

Now if we try to execute the preceding query then it will throw an error such as:

Insert Into Products\_View\_6 values ('Caramel',270)

**Output:**



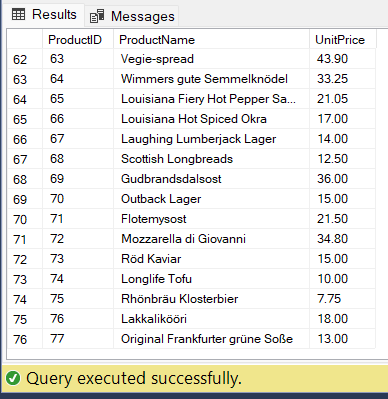
**DML Query in View**

In a view we can implement many types of DML query like insert, update and delete. But for a successful implementation of a DML query we should use some conditions like:

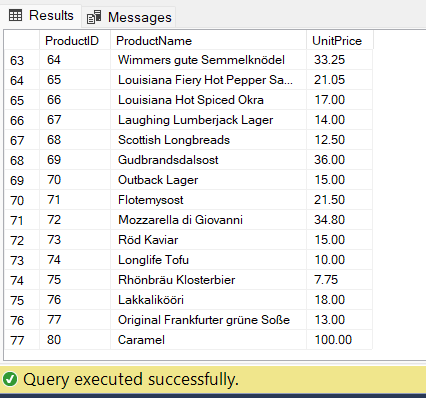
1. View should not contain multiple tables.
2. View should not contain set function.
3. View should not use the Distinct keyword.
4. View should not contain Group By, having clauses.
5. View should not contain Sub query.
6. View should not use Set Operators.
7. All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

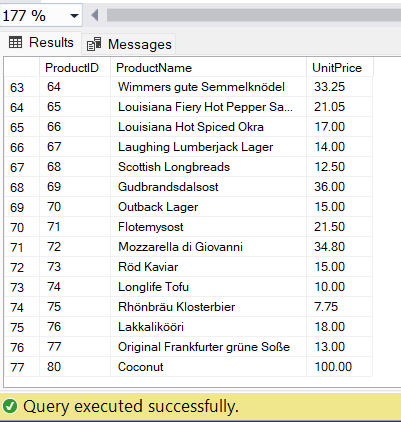
If we use the preceding conditions then we can implement a DML Query in the view without any problem. Let us see an example.

SELECT \* FROM Products\_View\_6



Now we implement a DML Query as in the following:

1. Insert Into Products\_View\_6 values ('Caramel',270)
2. Update Products\_View\_6 SET ProductName = 'Coconut' WHERE ProductId = 80;

****

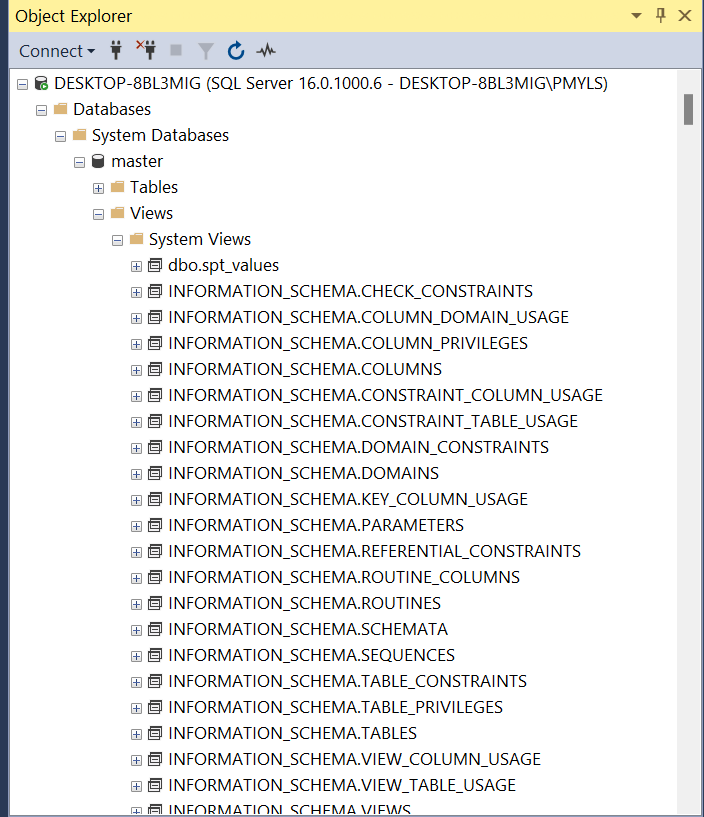
1. DELETE FROM Products\_View\_6 WHERE ProductId = 80

A screenshot of a computer

Description automatically generated

**System Define Views:**

SQL Server also contains various predefined databases like Tempdb, Master, temp. Each database has their own properties and responsibility. Master data is a template database for all other user-defined databases. A Master database contains many Predefine\_View that work as templates for other databases and tables. Master databases contain nearly 230 predefined views.



These predefined views are very useful to us. Mainly we divide system views into the following two parts.

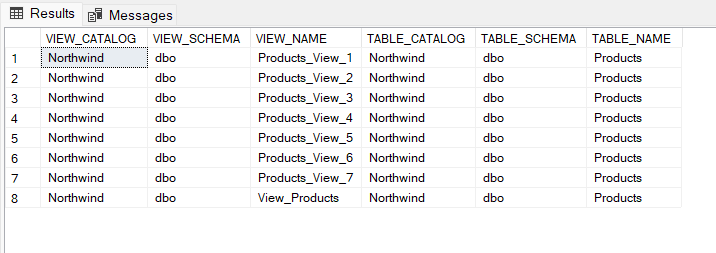
1. Information Schema
2. Catalog View

**Information schema:** There are nearly 21 Information Schemas in the System. These are used for displaying the most physical information of a database, such as table and columns. An Information Schema starts from INFORMATION\_SCHEMA.[View Name]. Let us see an example.

select \* from INFORMATION\_SCHEMA.VIEW\_TABLE\_USAGE

where TABLE\_NAME='Products'

**Output:**

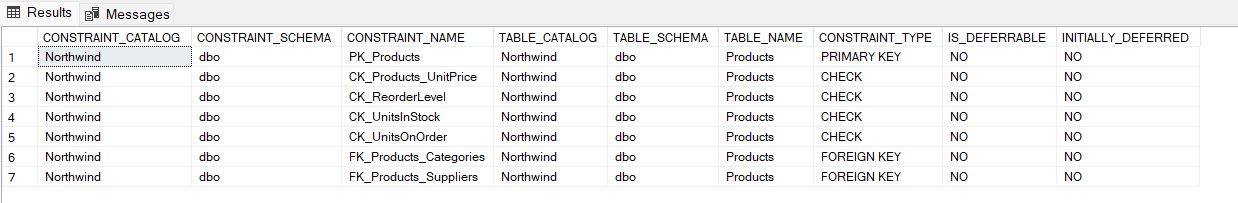
****

This Information\_Schema returns the details of all the views used by table Products.

select \* from INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS

where TABLE\_NAME='Products'

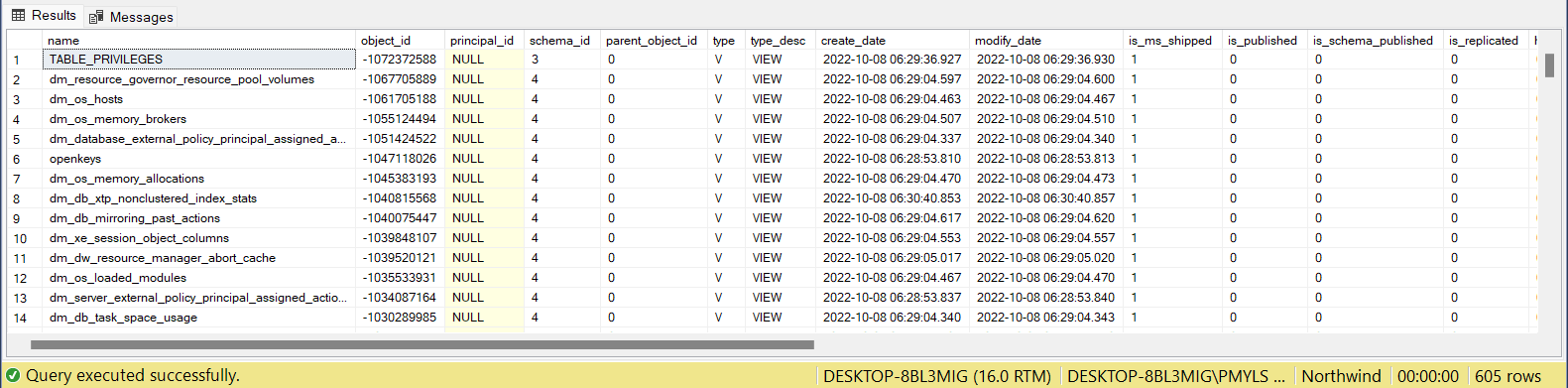
**Output:**



This **Information\_Schema** returns the information about the constraints of a table.

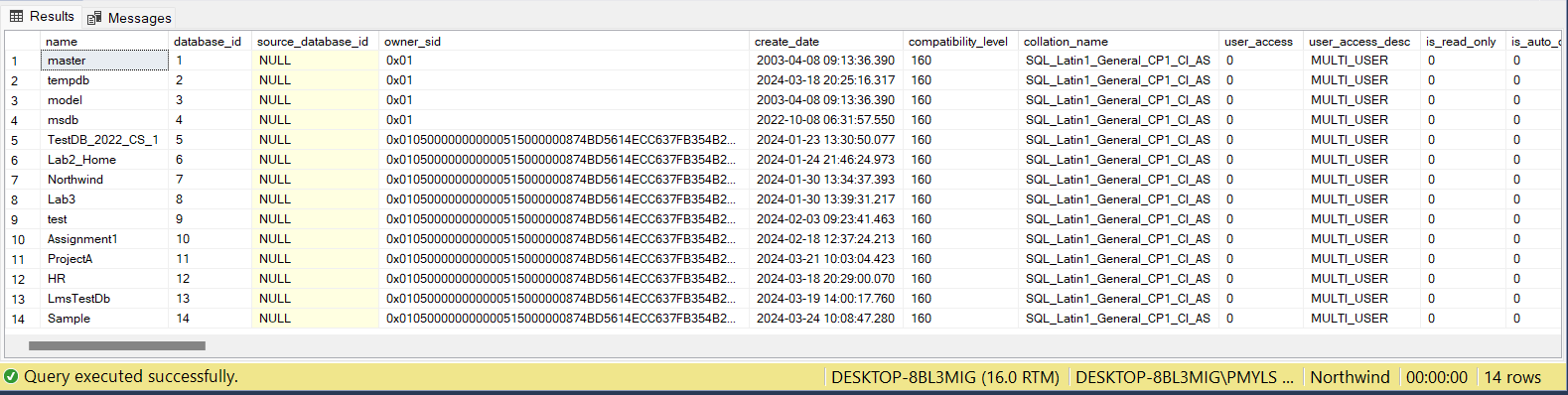
**Catalog View:** Catalog Views are categorized into various groups also. These are used to show the self-describing information of a database. These start with **“sys”**.

select \* from sys.all\_views



This query provides information to all types of views using a database.

select \* from sys.databases



This query will provide the information about all the databases defined by the system, including user-defined and system defined database.