



# ÍNDICE

<b>ÍNDICE</b>	<b>1</b>
<b>1. INSTALAR BASES DE DATOS DE EJEMPLO (DIFERENTES MODOS)</b>	<b>2</b>
1.1 PUBS DESDE SCRIPT	2
1.2 NORTHWIND CON ATTACH	4
1.3 ADVENTUREWORKS DESDE BACKUP	6
1.4 WIDEWORLDIMPORTERS CON BACPAC	8
<b>2. BASES DE DATOS CONTENIDAS</b>	<b>11</b>
<b>3. MAINTENANCE PLANS</b>	<b>15</b>
<b>4. FILESTREAM / FILETABLE</b>	<b>20</b>
FILESTREAM	20
FILETABLE	24
<b>5. OPERACIONES CON PARTICIONES</b>	<b>27</b>
5.1 SPLIT	33
5.2 MERGE	34
5.3 SWITCH	35
5.4 TRUNCATE	37
<b>6. TABLAS TEMPORALES / VERSIÓN DEL SISTEMA</b>	<b>37</b>
<b>7. TABLAS IN MEMORY</b>	<b>44</b>

# 1. INSTALAR BASES DE DATOS DE EJEMPLO (DIFERENTES MODOS)

[Volver al índice →](#)

Las siguientes bases de datos por lo general son usadas con fines de aprendizaje, (comunidad, blogs, etc) y con ellas realizaremos diversos métodos de instalación.

1. Northwind
2. Pubs
3. AdventureWorks
4. Wide World Importers

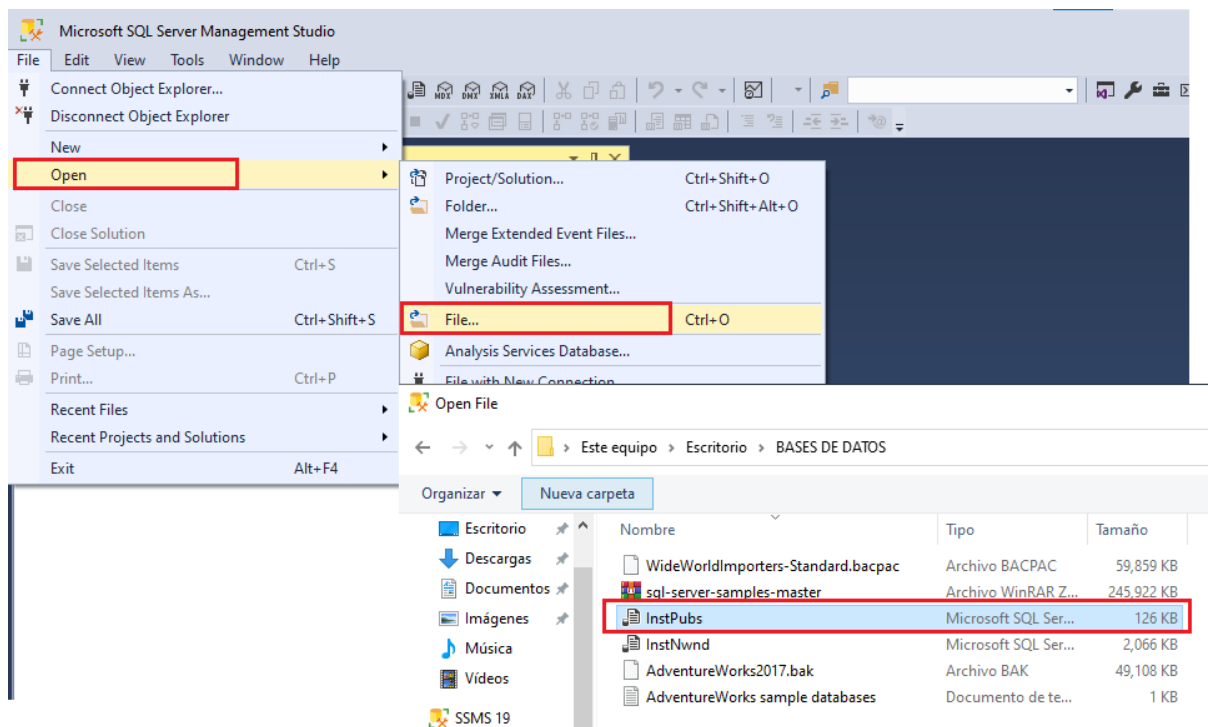
Para crear estas BD lo podemos hacer mediante diferentes métodos:

- Ejecutando un script. (archivo con extensión .sql)
- Restaurando un backup. (archivo con extensión .bak)
- Importando un archivo desde la nube (extensión .bacpac)

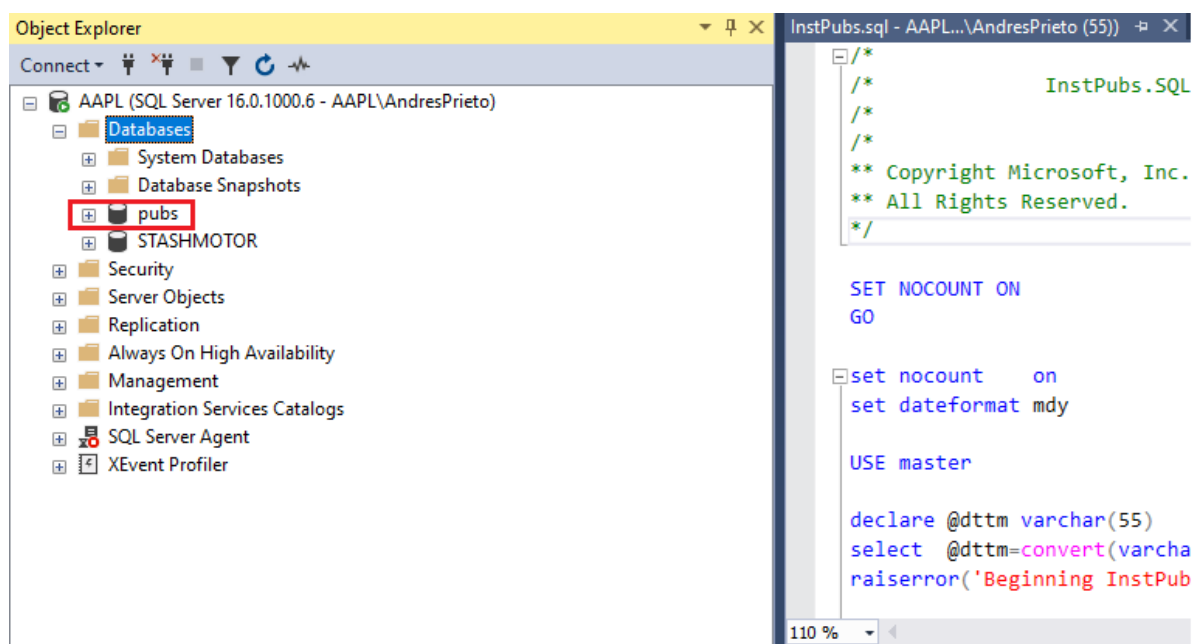
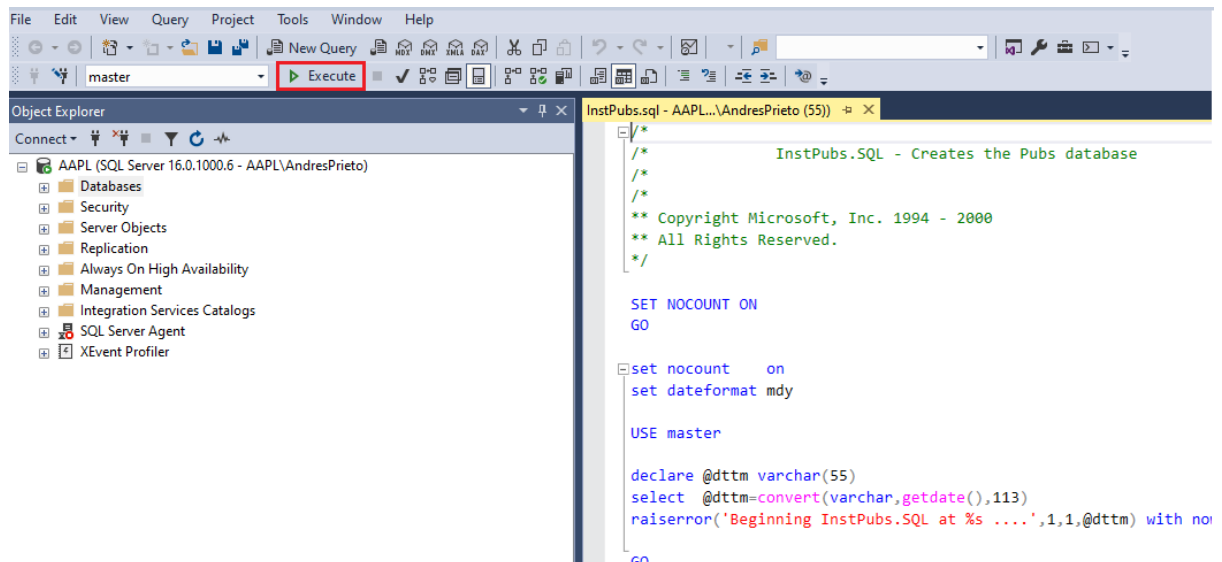
## 1.1 PUBS DESDE SCRIPT

[Volver al índice →](#)

Para ello abrimos el fichero con la extensión .sql desde ssms , tomando como ejemplo pubs.sql



Ahora procedemos a ejecutar el script y verificamos la correcta creación de la base de datos.

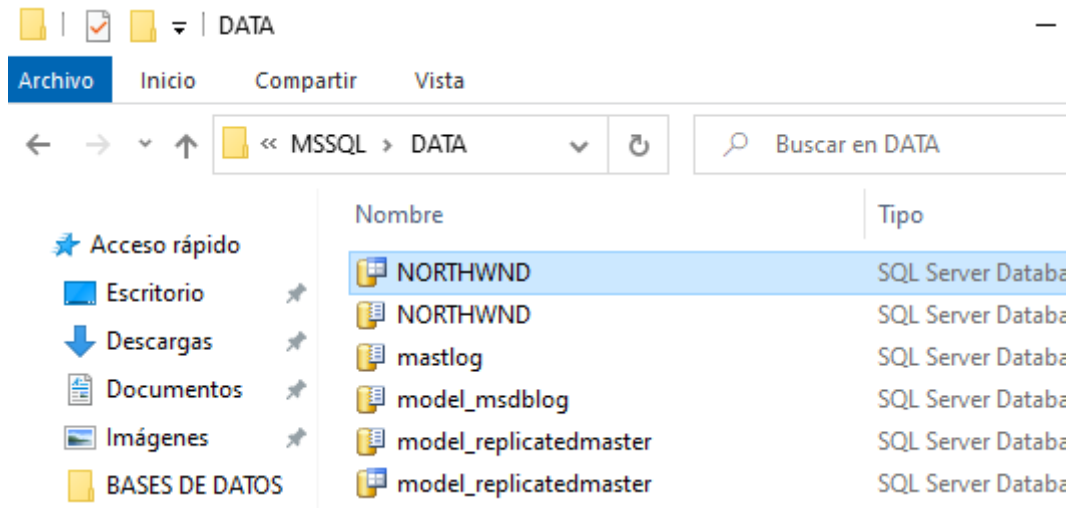


## 1.2 NORTHWIND CON ATTACH

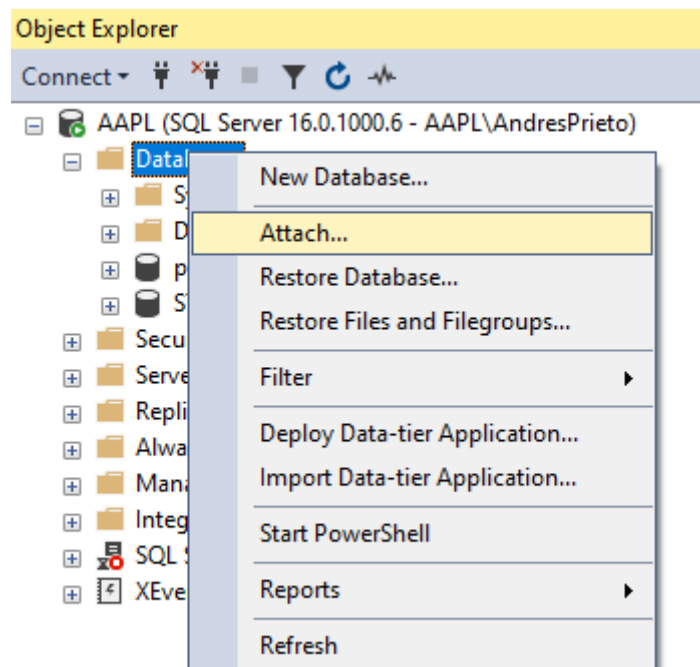
[Volver al índice →](#)

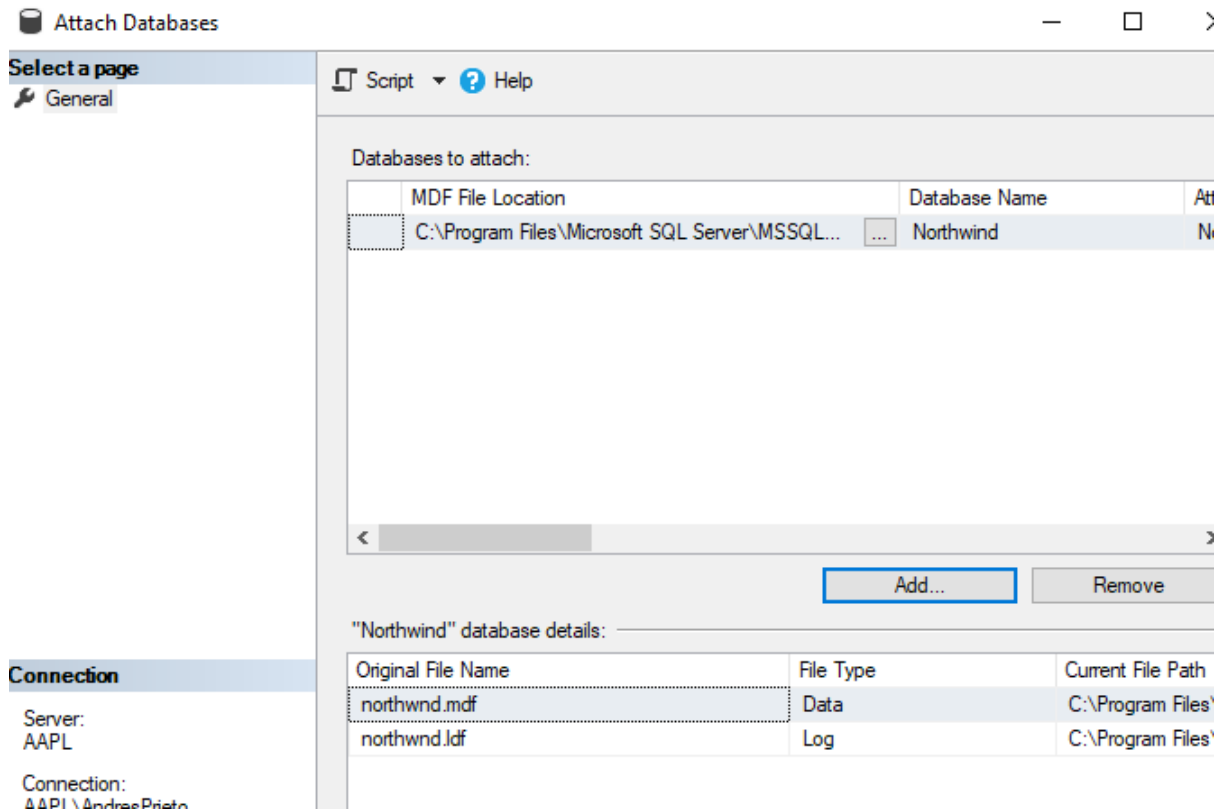
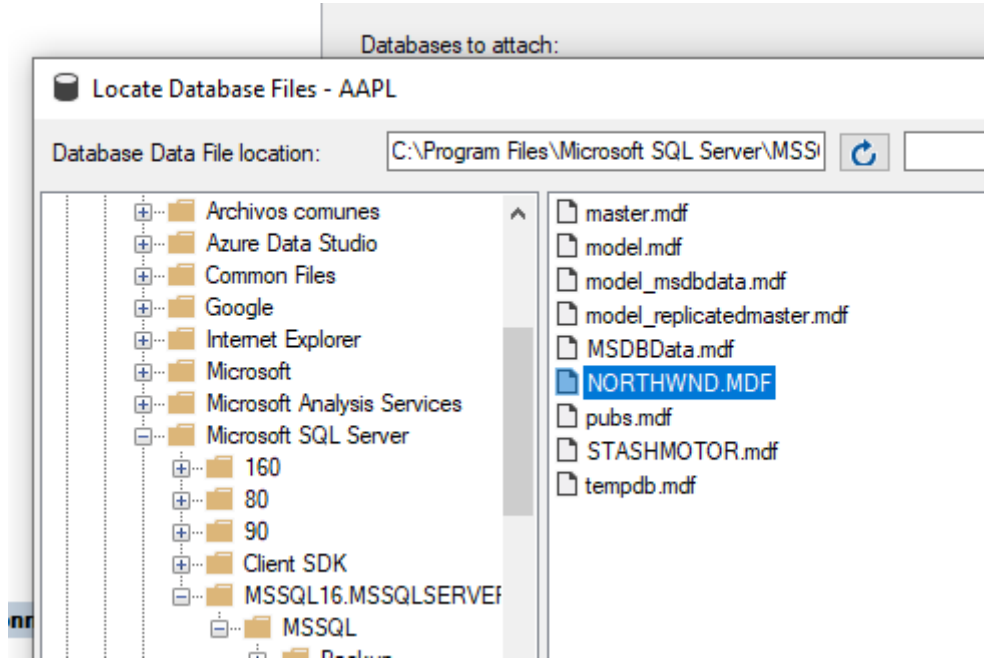
Generalmente los ficheros con extensión .mdf / .ldf se ubican en la siguiente ruta:

C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA

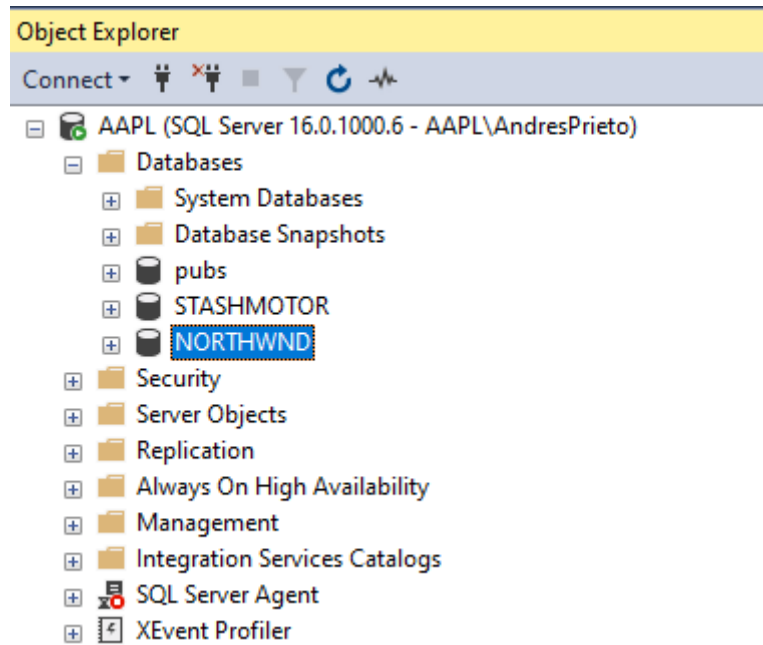


Realizamos los siguientes pasos





Una vez seleccionamos el fichero y damos click en continuar, verificamos que se haya creado correctamente la base de datos

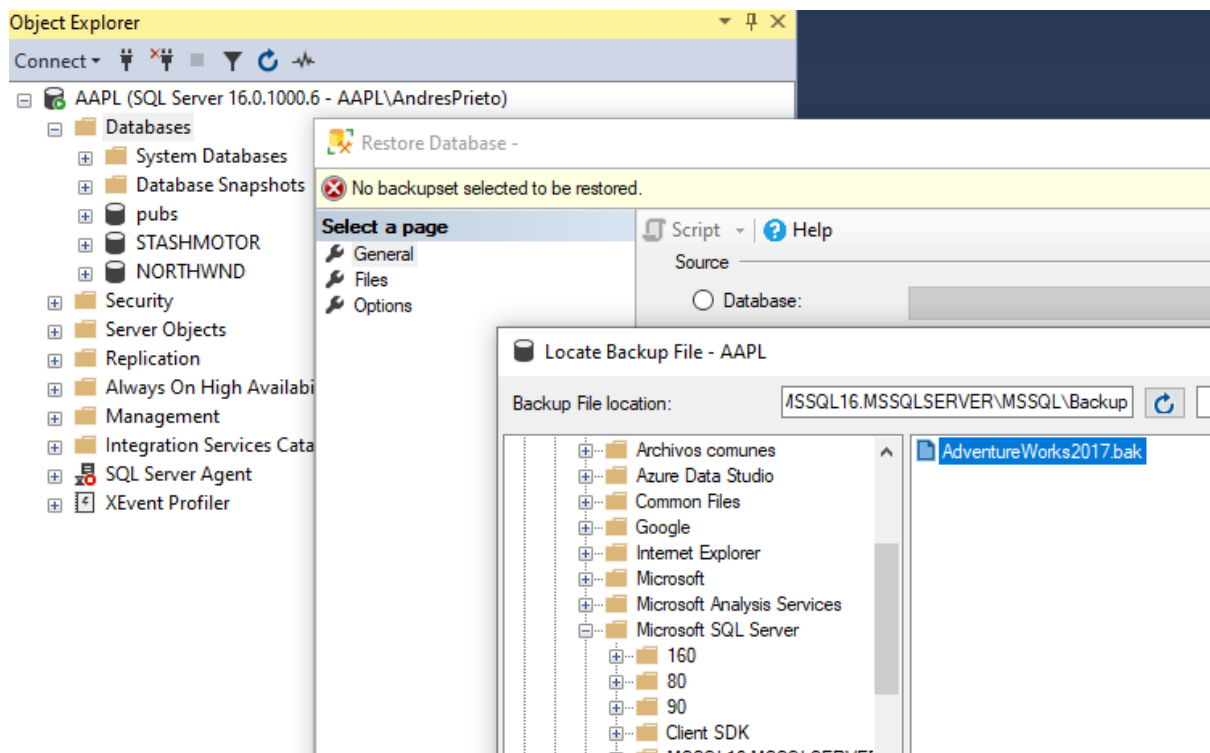


## 1.3 ADVENTUREWORKS DESDE BACKUP

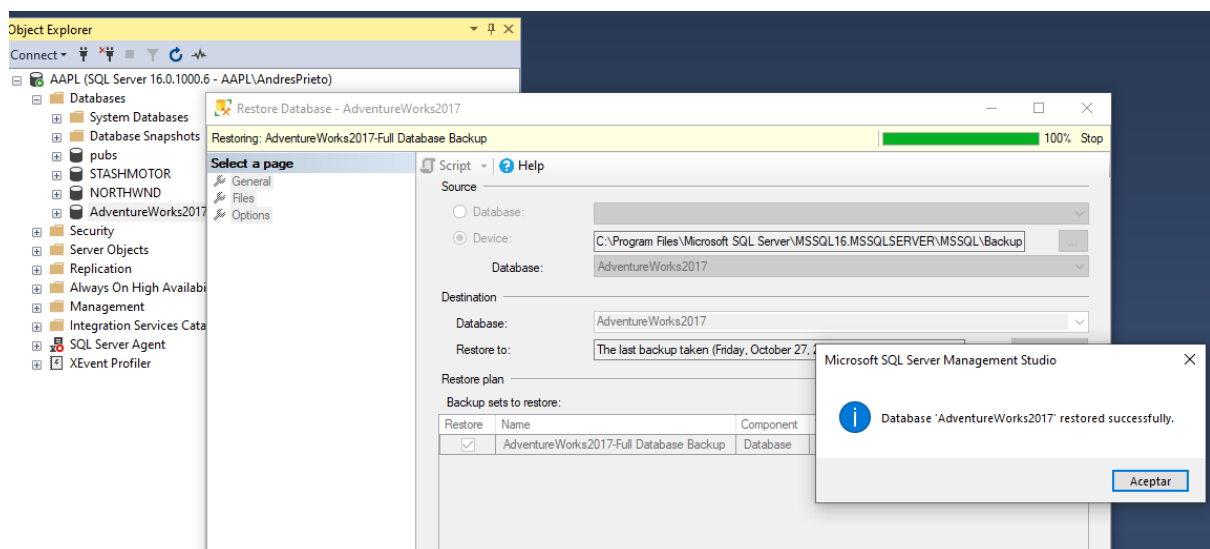
[Volver al índice →](#)

Generalmente y por buena práctica debemos tener centralizados todos los backup en una misma carpeta, usualmente dicha carpeta se encuentra en la siguiente ruta:

C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\Backup

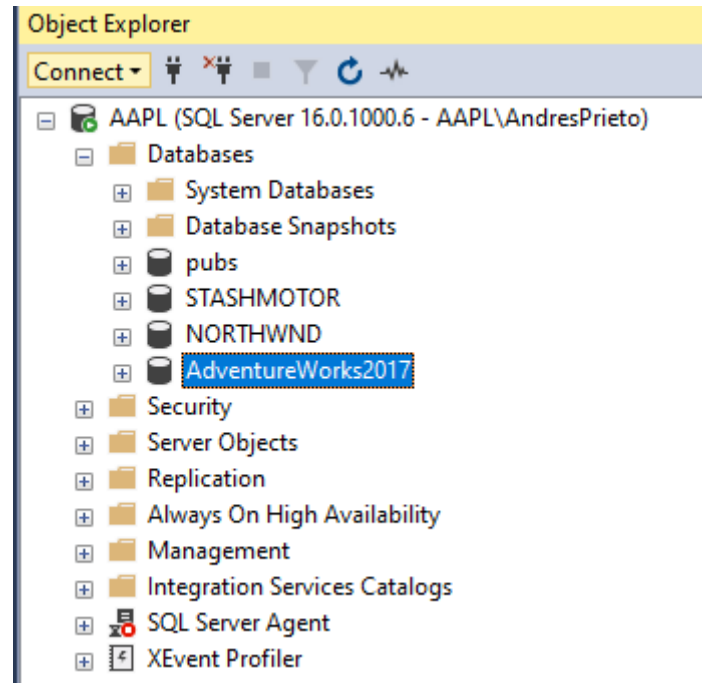


Seleccionamos el archivo de backup de la base de datos.





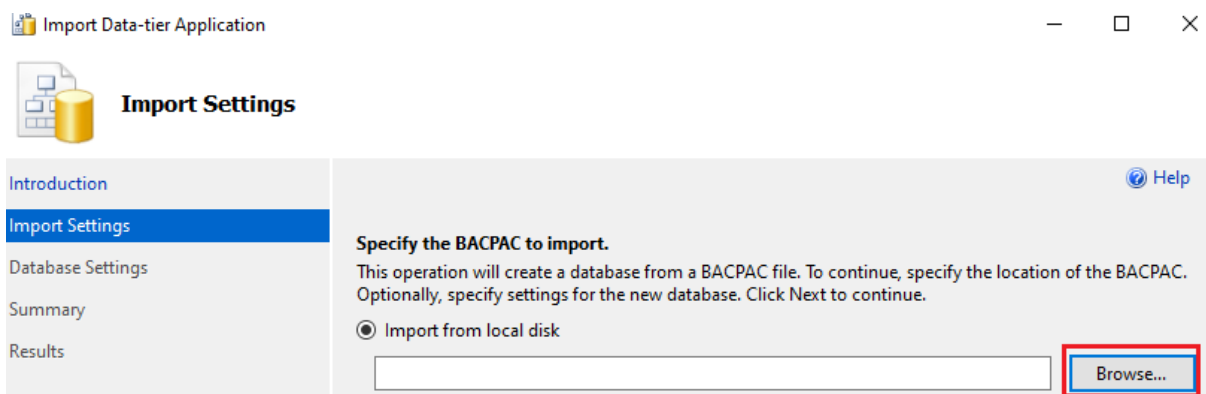
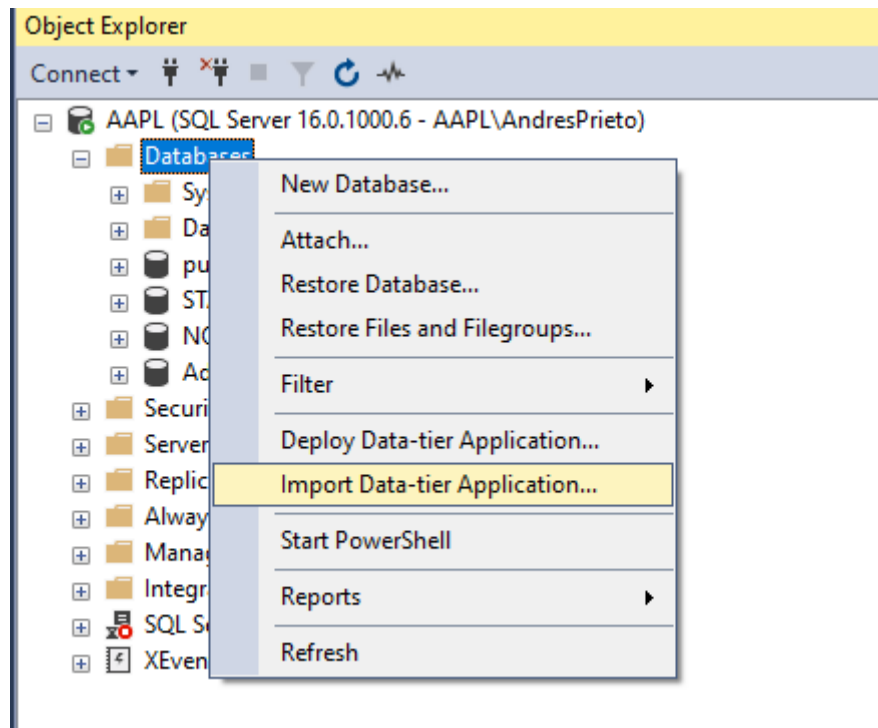
Una vez finalizado el proceso de restauración procedemos a verificar que fue procesado de manera exitosa.

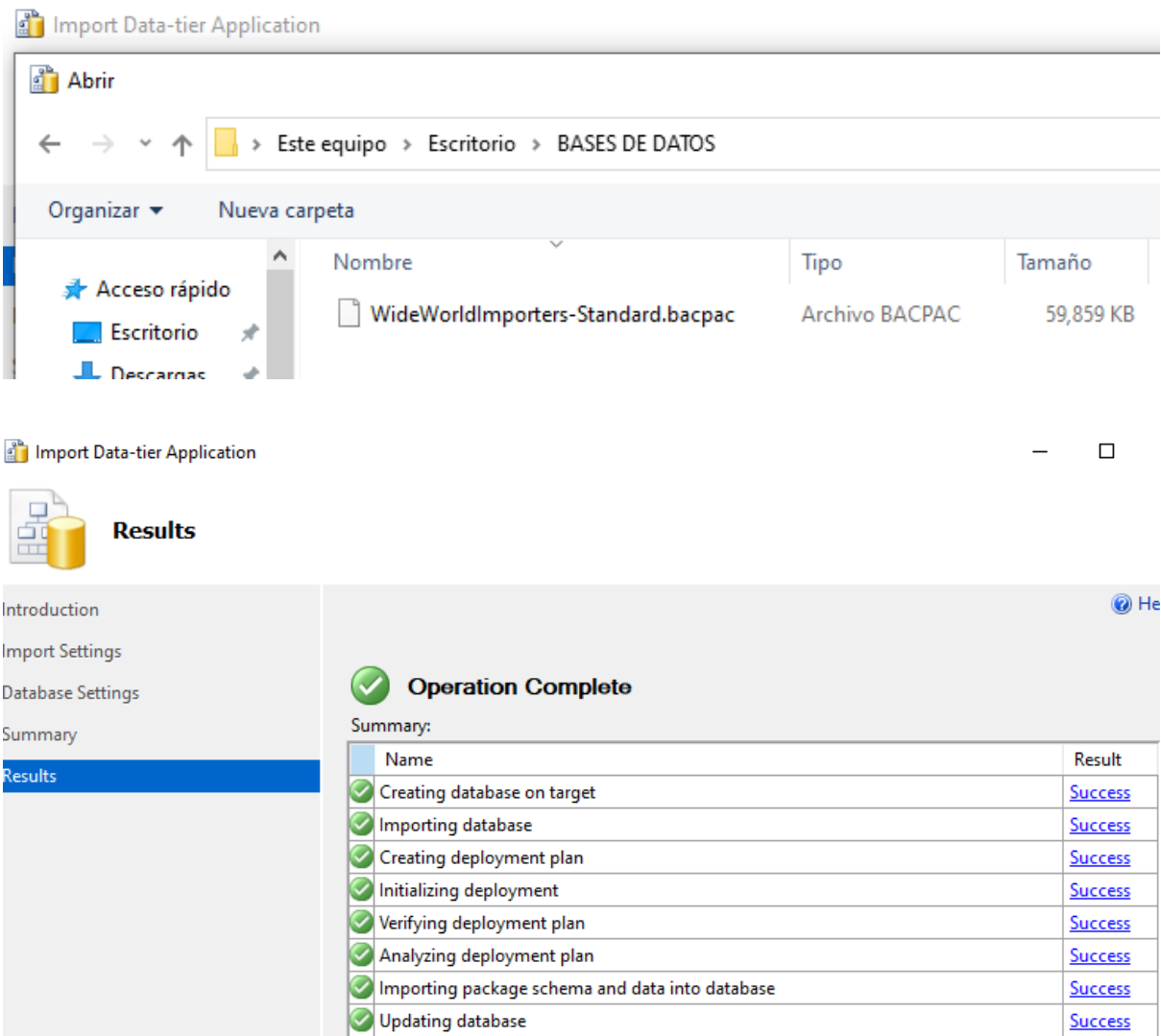


## 1.4 WIDEWORLDIMPORTERS CON BACPAC

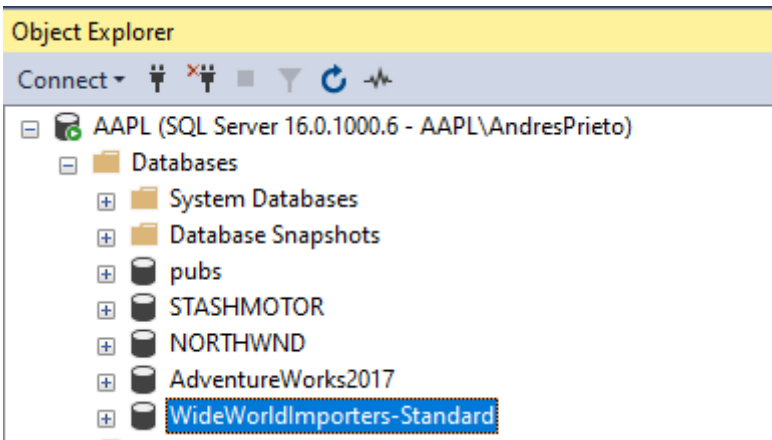
[Volver al índice →](#)

Los archivos tipo bacpac son utilizados para comunicaciones entre servidores locales y servidores en la nube. Para ello debemos realizar los siguientes pasos:





Verificamos la correcta importación de la base de datos.



## 2. BASES DE DATOS CONTENIDAS

[Volver al índice →](#)

Una base de datos contenida (en inglés "contained database") es una base de datos que es autónoma e independiente, y que tiene todas sus dependencias, metadatos y configuraciones dentro de la propia base de datos. Esto significa que una base de datos contenida no depende de la instancia del servidor de base de datos en la que se ejecuta y puede ser copiada y movida fácilmente de un servidor a otro.

En una base de datos contenida, los usuarios y roles son definidos y administrados dentro de la base de datos misma, en lugar de ser definidos en el nivel del servidor de base de datos. Además, las opciones de configuración como el nivel de compatibilidad de la base de datos y los valores de configuración del servidor de base de datos son controlados a nivel de la base de datos.

Las bases de datos contenidas son una opción popular para aplicaciones que necesitan ser desplegadas en entornos de múltiples servidores o para aplicaciones de nube, ya que facilitan el movimiento y la gestión de la base de datos en diferentes ubicaciones.

Para ello crearemos una base de datos contenida, tomando como referencia nuestro proyecto STASHMOTOR.

```
-- Inicialmente nos aseguramos de estar en master--
```

```
USE MASTER  
GO
```

```
-- Primero activamos las opciones avanzadas--
```

```
EXEC sp_configure 'show advanced options', 1  
GO
```

```
--Actualizamos el valor--
```

```
RECONFIGURE  
GO
```

```
--Activamos la característica--
```

```
EXEC sp_configure 'contained database authentication' , 1  
GO
```

--Ya hemos preparado el entorno para las siguientes ejecuciones--

--Activamos la base de datos y configuramos la misma como contenida--

```
CREATE DATABASE STASHMOTOR_CONTENIDA
GO
```

```
ALTER DATABASE STASHMOTOR_CONTENIDA
SET RESTRICTED_USER
WITH ROLLBACK IMMEDIATE;
GO
```

```
ALTER DATABASE STASHMOTOR_CONTENIDA
SET containment=partial;
GO
```

```
ALTER DATABASE STASHMOTOR_CONTENIDA
SET MULTI_USER;
GO
```

```
USE STASHMOTOR_CONTENIDA
GO
```

--Creamos el usuario Prieto dentro de la base de datos--

```
DROP USER IF EXISTS Prieto
GO
```

```
CREATE USER Prieto
    WITH PASSWORD= '1234.',
    DEFAULT_SCHEMA=[dbo]
GO
```

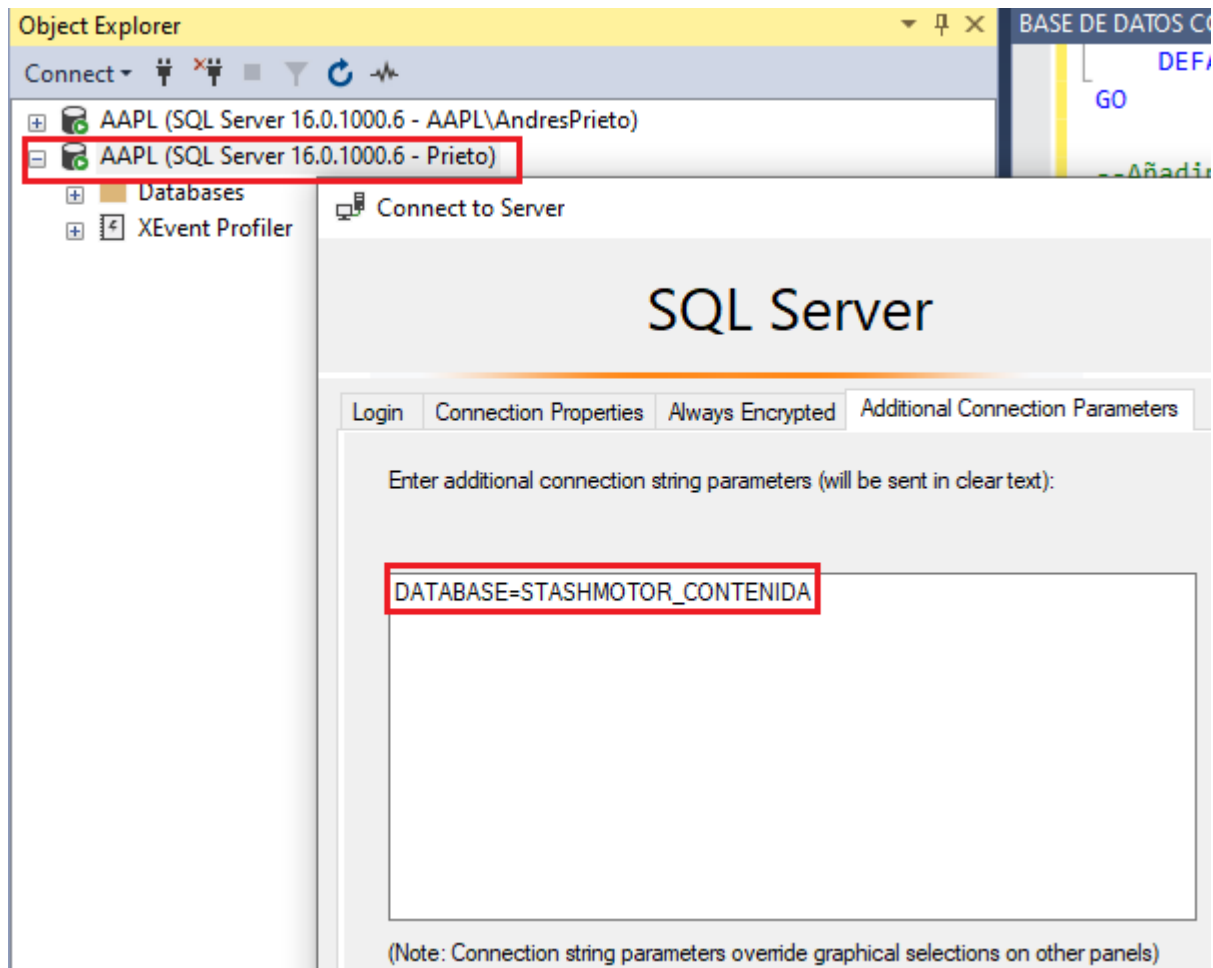
--Añadimos el usuario que acabos de crear al rol owner--

```
ALTER ROLE db_owner
    ADD MEMBER Prieto
GO
```

--Otorgamos permisos de "grant" al usuario para que pueda conectares a la base de datos--

```
GRANT CONNECT TO Prieto
GO
```

Ahora intentamos conectarnos con el usuario creado, para ello debemos asegurarnos previamente tener activa la autenticación mixta.



Ya estando conectados con el usuario Prieto, procedemos a crear una tabla de ejemplo.

--Una vez conectados procedemos a realizar la creacion de una tabla--

```
CREATE TABLE [dbo].[ArticulosContenida](
    [SKU] VARCHAR (10) NULL,
    [Nombre] VARCHAR (20) NULL,
    [Descripcion] VARCHAR (50)
) ON [PRIMARY]
GO
```

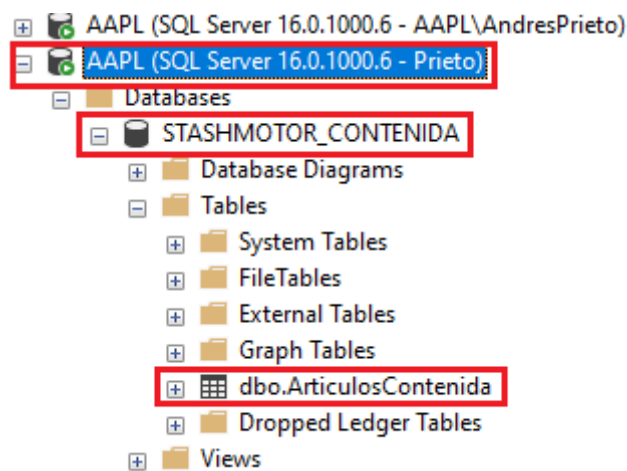
110 %

Messages

Commands completed successfully.

Completion time: 2023-02-16T04:02:56.1247942+01:00

```
CREATE TABLE [dbo].[ArticulosContenida](
    [SKU] VARCHAR (10) NULL,
    [Nombre] VARCHAR (20) NULL,
    [Descripcion] VARCHAR (50)
) ON [PRIMARY]
GO
```



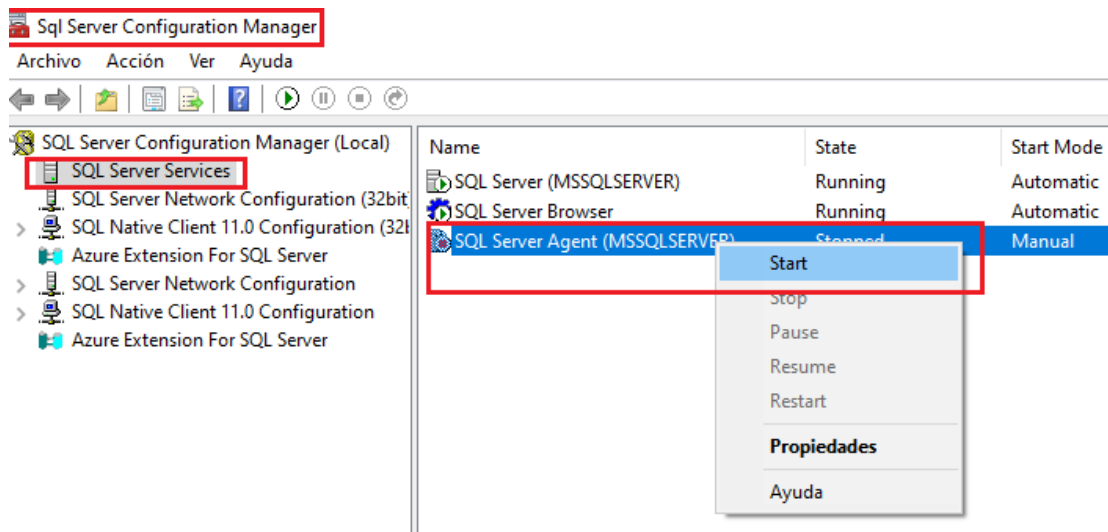
### 3. MAINTENANCE PLANS

[Volver al índice →](#)

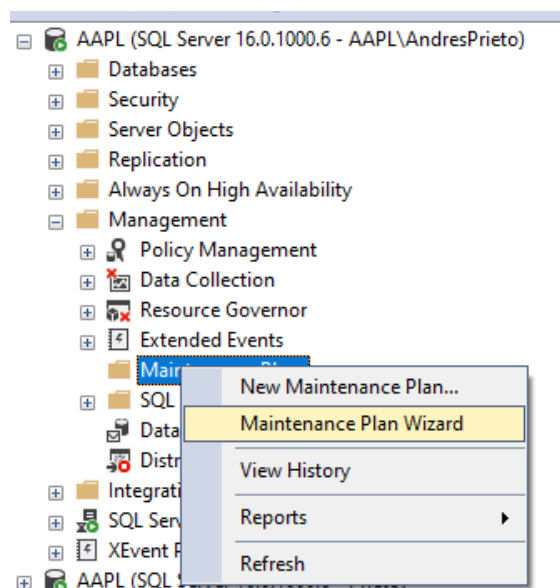
Esta herramienta es capaz de realizar diversas tareas de administración de bases de datos, pero haremos uso de ella en este caso para realizar backups programados de manera periódica y automatizada.

Inicialmente debemos comprobar que se encuentra en funcionamiento el servicio de “sql server agent” ya que usualmente se encuentra deshabilitado.

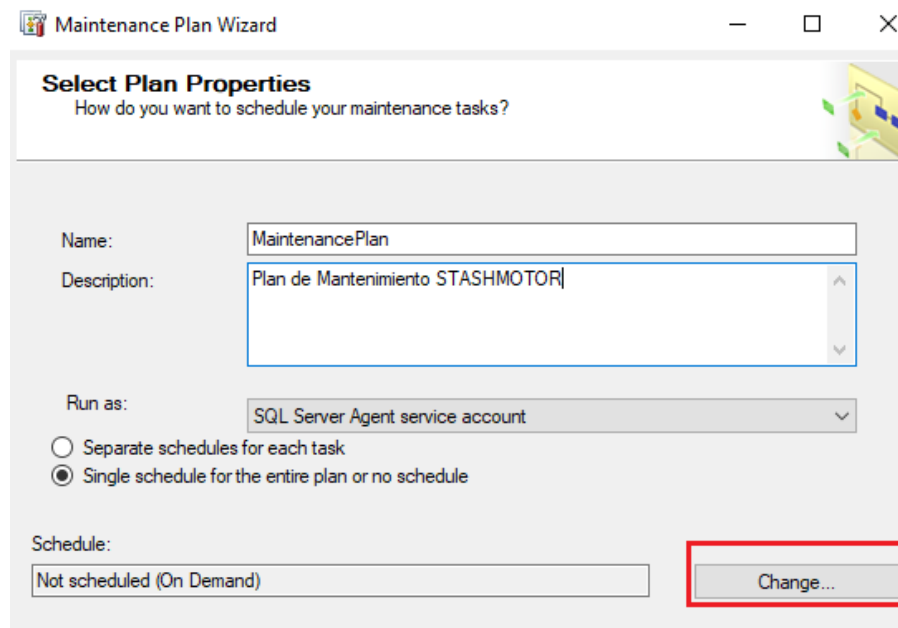
Para ello haremos lo siguiente:



Una vez tenemos el servicio en funcionamiento nos dirigimos al siguiente apartado







**Maintenance Plan Wizard**

**Select Plan Properties**  
How do you want to schedule your maintenance tasks?

Name: MaintenancePlan

Description: Plan de Mantenimiento STASHMOTOR

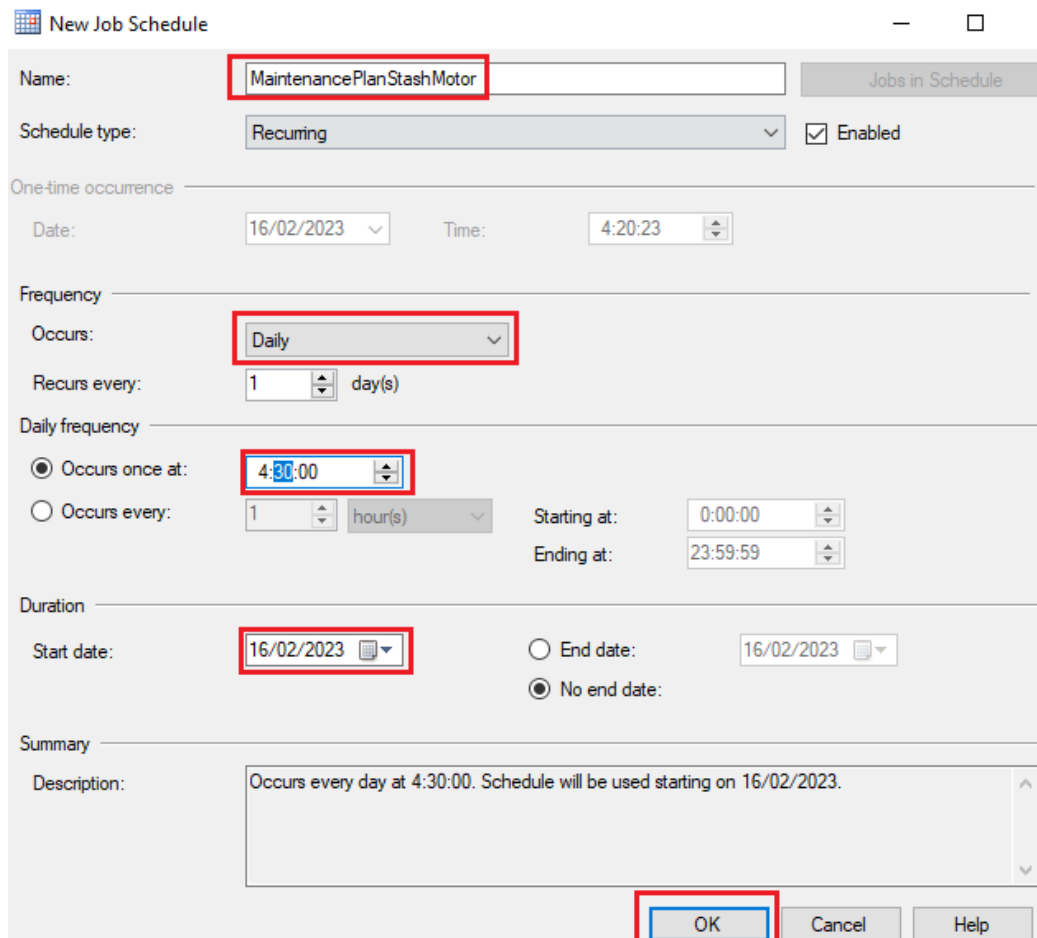
Run as: SQL Server Agent service account

☐ Separate schedules for each task

☒ Single schedule for the entire plan or no schedule

Schedule: Not scheduled (On Demand) Change...

Con fines demostrativos, hemos indicado que deseamos el backup todos los días a las 4:30 am



**New Job Schedule**

Name: MaintenancePlanStashMotor

Schedule type: Recurring

☒ Enabled

One-time occurrence

Date: 16/02/2023 Time: 4:20:23

Frequency

Occurs: Daily

Recurs every: 1 day(s)

Daily frequency

☒ Occurs once at: 4:30:00

☐ Occurs every: 1 hour(s)

Starting at: 0:00:00

Ending at: 23:59:59

Duration

Start date: 16/02/2023

☐ End date: 16/02/2023

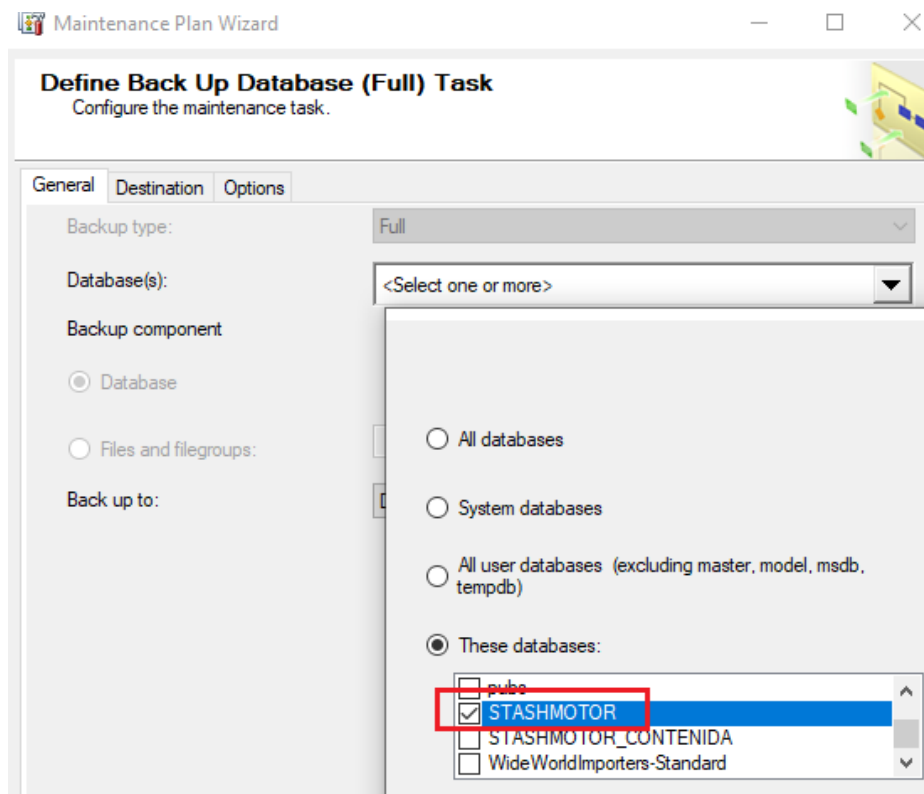
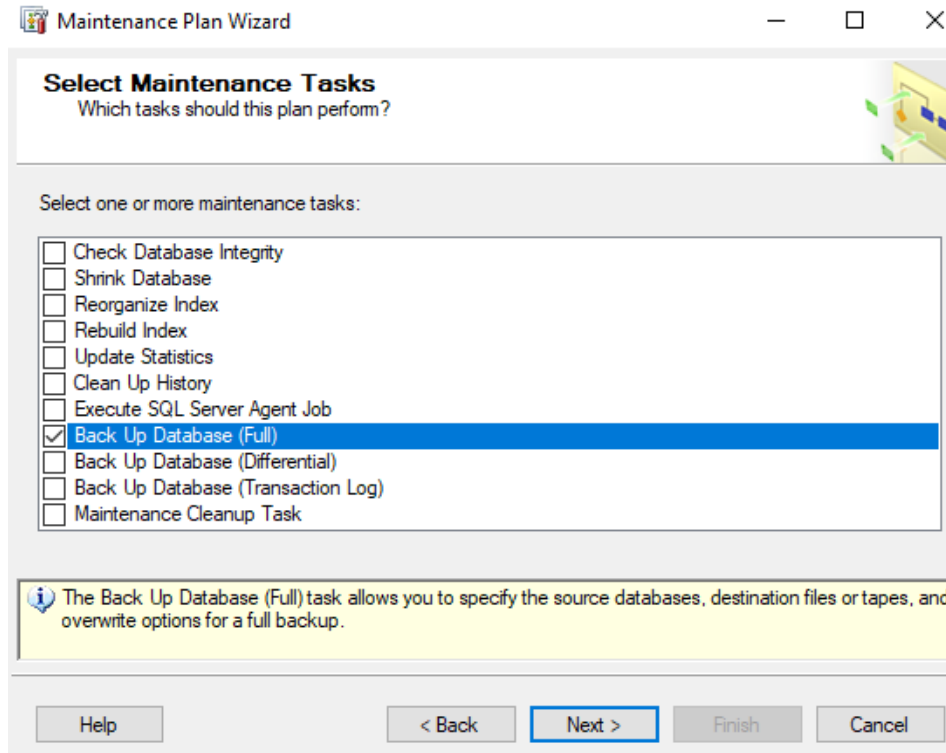
☒ No end date:

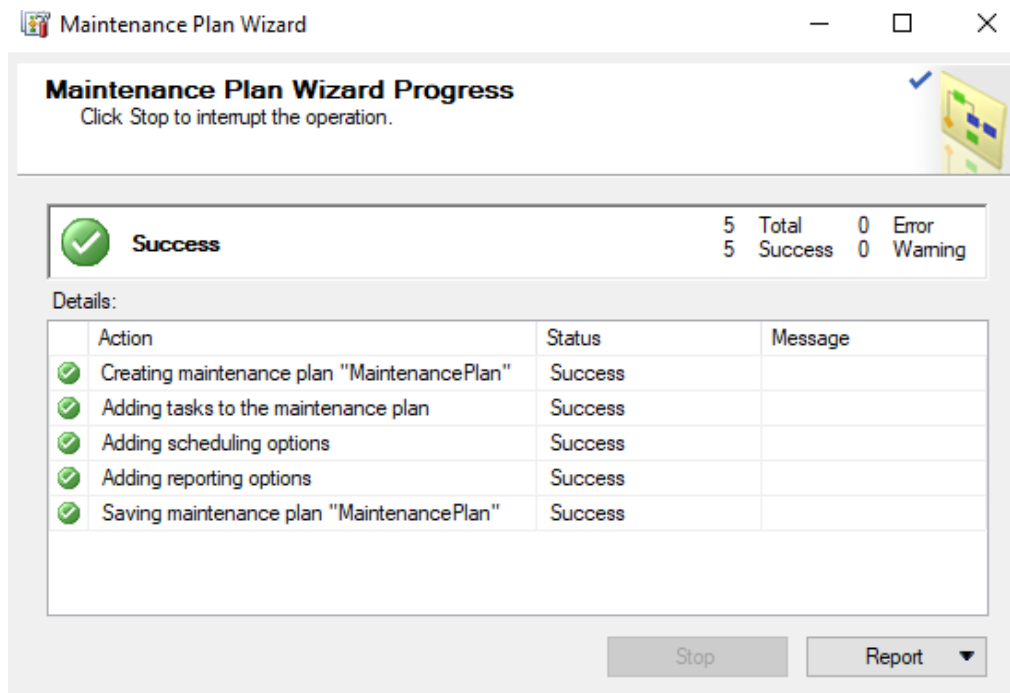
Summary

Description: Occurs every day at 4:30:00. Schedule will be used starting on 16/02/2023.

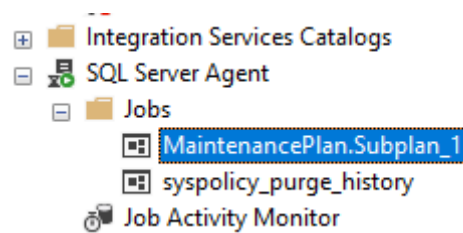
OK Cancel Help

Nos permite seleccionar el tipo de backup que deseamos, que en este caso hemos elegido el tipo de backup full, pero podría ser cualquier de las otras opciones dependiendo de las necesidades.

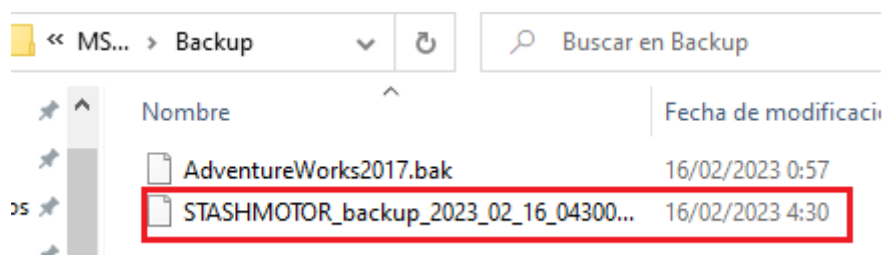




Verificamos que se ha creado correctamente el "job"



Y para demostrarlo verificamos que se ha realizado el backup con la hora indicada.



## 4. FILESTREAM / FILETABLE

[Volver al índice →](#)

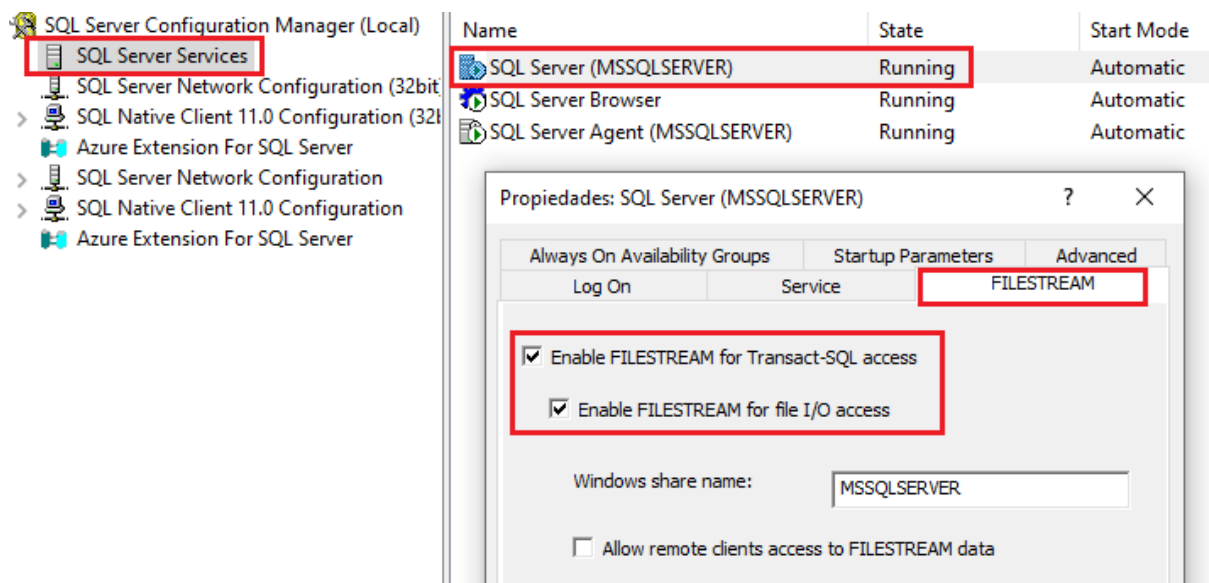
### FILESTREAM

Es una característica que nos permite almacenar y administrar grandes objetos binarios de manera eficiente, en un sistema de archivos en lugar de almacenarlos directamente en la base de datos, lo destacable en este caso es que en caso de backups las imágenes igual serán almacenadas en el mismo.

Teniendo en cuenta que en nuestro proyecto contamos con una tabla de artículos y nos interesa almacenar una imagen de cada uno de ellos, hemos de ir insertando los mismos acompañado de su imagen.

Primero debemos habilitar el filestream y podemos hacerlo de donde maneras:

#### ENTORNO GRÁFICO



## TRANSACTION

```
USE STASHMOTOR
```

```
GO
```

```
EXEC sp_configure filestream_access_level,2
```

```
RECONFIGURE
```

```
GO
```

```
ALTER DATABASE STASHMOTOR
```

```
    ADD FILEGROUP [PRIMARY_FILESTREAM]
```

```
    CONTAINS FILESTREAM
```

```
GO
```

```
--AÑADIR LA CARPETA AL FILEGROUP--
```

```
ALTER DATABASE STASHMOTOR
```

```
    ADD FILE (
```

```
        NAME= 'STASHMOTOR_FILESTREAM',
```

```
        FILENAME = 'C:\Program Files\Microsoft SQL  
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\FILESTREAM' )
```

```
    TO FILEGROUP [PRIMARY_FILESTREAM]
```

```
GO
```

```
--AÑADIMOS DOS NUEVOS CAMPOS EN LA TABLA ARTICULO PARA REALIZAR LA  
DEMOSTRACION--
```

```
ALTER TABLE Articulo
```

```
    ADD ArticuloID UNIQUEIDENTIFIER ROWGUIDCOL NOT NULL UNIQUE
```

```
GO
```

```
ALTER TABLE Articulo
```

```
    ADD Imagen VARBINARY(MAX) FILESTREAM
```

```
GO
```

```
SELECT [SKU],[Nombre],[Descripcion],[PVP],[Imagen] FROM Articulo
```

```
GO
```

```
--INSERTAMOS LOS CAMPOS E IMAGENES--
```

```
INSERT INTO Articulo (SKU, Nombre, Descripcion, PVP,
```

```
Inventario_ID_Inventario, Detalle_Venta_ID_DV, Detalle_Compra_ID_DC,  
Modelo_ID_Modelo, ArticuloID, Imagen)
```

```
VALUES ( 'SKU1', 'Llave de Gasolina', 'Regula la salida de gasolina',  
35.00, 1, 1, 1, 1, NEWID(),
```

```
(SELECT * FROM OPENROWSET (BULK 'C:\ImagenArticulos\SKU1.PNG',
```

```
SINGLE_BLOB) AS f ));
GO
```

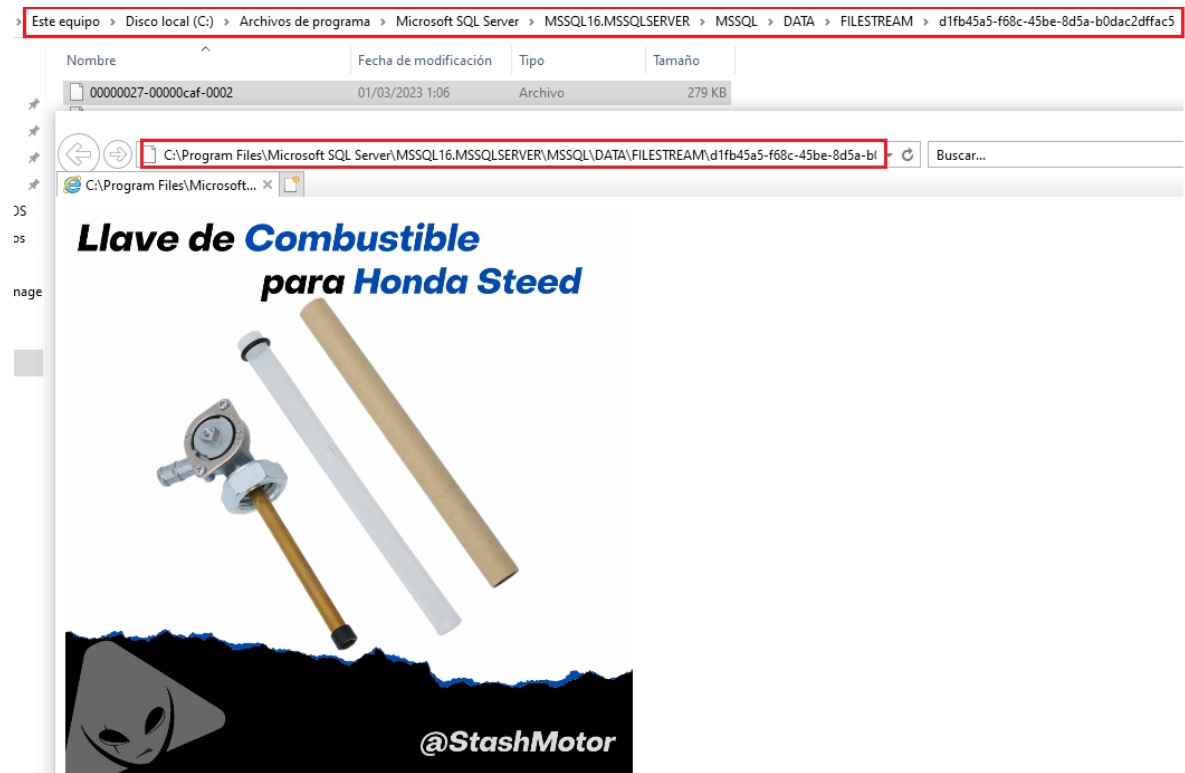
```
INSERT INTO Articulo (ArticuloID, SKU, Nombre, Descripcion, PVP, Imagen,
Inventario_ID_Inventario, Detalle_Venta_ID_DV, Detalle_Compra_ID_DC,
Modelo_ID_Modelo)
VALUES (NEWID(), 'SKU2', 'Filtro de Aire', 'Regula la entrada de aire',
25.00,
(SELECT * FROM OPENROWSET (BULK 'C:\ImagenArticulos\SKU2.PNG',
SINGLE_BLOB) AS f), 1, 1, 1, 1);
GO
```

```
INSERT INTO Articulo (ArticuloID, SKU, Nombre, Descripcion, PVP, Imagen,
Inventario_ID_Inventario, Detalle_Venta_ID_DV, Detalle_Compra_ID_DC,
Modelo_ID_Modelo)
VALUES (NEWID(), 'SKU3', 'Kit Carburador', 'Kit', 38.00,
(SELECT * FROM OPENROWSET (BULK 'C:\ImagenArticulos\SKU3.PNG',
SINGLE_BLOB) AS f), 1, 1, 1, 1);
GO
```

Al seleccionar los campos de la tabla obtenemos un resultado como este, el cual se observan las imágenes almacenadas de forma binaria

Results					
Messages					
	SKU	Nombre	Descripcion	PVP	Imagen
1	SKU1	Llave de Gasolina	Regula la salida de gasolina	35	0x89504E470D0A1A0A0000000D4948445200000438000004...
2	SKU2	Filtro de Aire	Regula la entrada de aire	25	0x89504E470D0A1A0A0000000D4948445200000438000004...
3	SKU3	Kit Carburador	Kit	38	0x89504E470D0A1A0A0000000D4948445200000438000004...

Podemos comprobar que dichas imágenes se encuentran guardadas en el directorio de filestream



## FILETABLE

[Volver al índice →](#)

En este caso con fines de simplificar la demostración, crearemos una base de datos "STASHMOTORFILETABLE".

--HABILITAMOS FILESTREAM Y REINICIAMOS EL SERVICIO--

```
EXEC sp_configure filestream_access_level, 2
RECONFIGURE
GO
```

--COMPROBAMOS LA EXISTENCIA DE LA BASE DE DATOS--

```
DROP DATABASE IF EXISTS STASHMOTORFILETABLE
GO
```

--CREAMOS LA BASE DE DATOS--

```
CREATE DATABASE STASHMOTORFILETABLE
ON PRIMARY
(
    NAME = STASHMOTOR_DATOS,
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\FileTable\stashmotor.mdf'
),
FILEGROUP FileStreamFG CONTAINS FILESTREAM
(
    NAME = SQLFileTable,
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\FileTable\Contenedor_filetable'
)
LOG ON
(
    NAME = SQLFileTable_Log,
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\FileTable\log_filetable.ldf'
)
WITH FILESTREAM
(
    NON_TRANSACTED_ACCESS = FULL,
    DIRECTORY_NAME = 'Contenedor_filetable'
);
```



GO

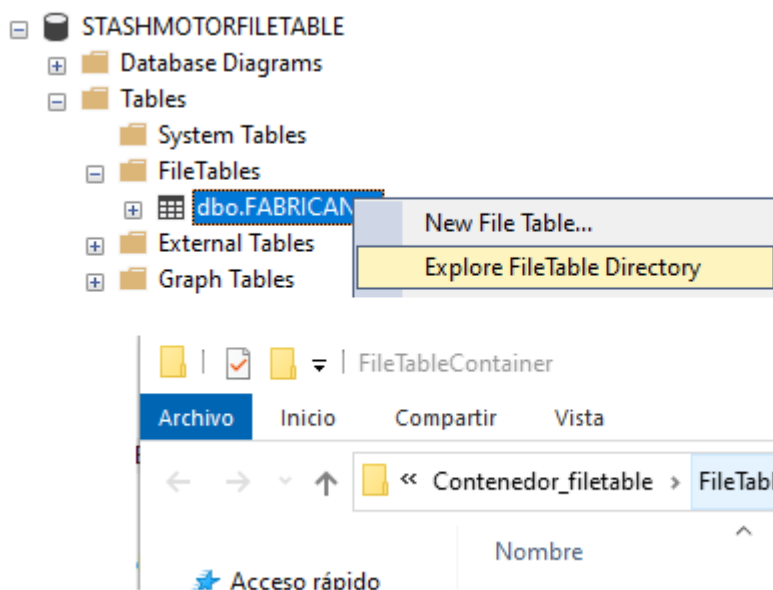
El cual luego de haber realizado el script nos creará una estructura dentro de la carpeta FILETABLE.

> Este equipo > Disco local (C:) > Archivos de programa > Microsoft SQL Server > MSSQL16.MSSQLSERVER > MSS				
	Nombre	Fecha de modificación	Tipo	Tamaño
	Contenedor_filetable	03/03/2023 0:04	Carpeta de archivos	
	log_filetable	03/03/2023 0:04	SQL Server Databa...	8.192 KB
	stashmotor	03/03/2023 0:04	SQL Server Databa...	8.192 KB

Luego creamos una tabla, de tipo filetable

```
CREATE TABLE FABRICANTE
AS FILETABLE
WITH
(
    FileTable_Directory = 'FileTableContainer',
    FileTable_Collate_Filename = database_default
);
GO
```

Procedemos a verificar su correcta creación desde el entorno grafico así como su path donde se guardarán los archivos de imágenes en el propio sistema.



Hacemos un select sobre la tabla y verificamos que la misma se encuentra vacía.

```
--HAREMOS UN SELECT SOBRE DICHA TABLA PUDIENDO VERIFICAR QUE LA MISMA--
--SE ENCUENTRA VACIA--

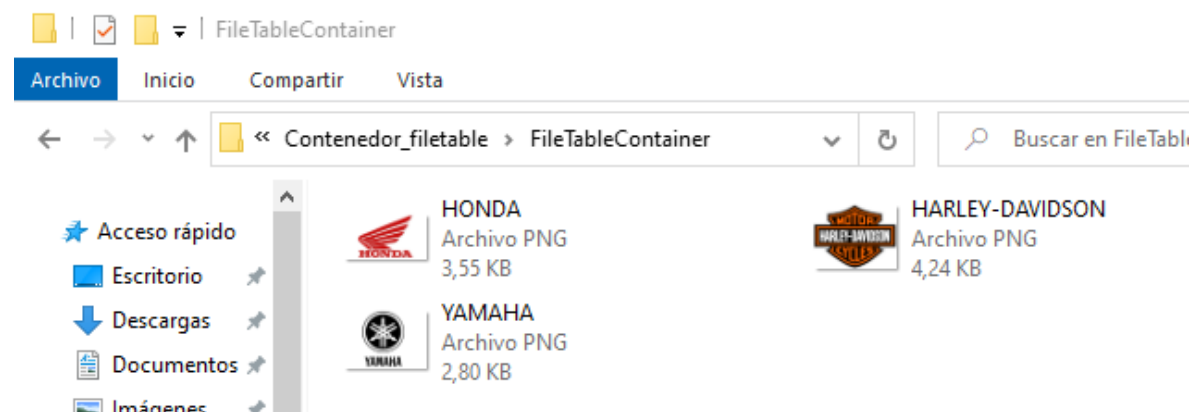
SELECT * FROM FABRICANTE
GO
```

%

Results Messages

stream_id	file_stream	name	path_locator	parent_path_locator	file_type	cached_file_size	creation_time
-----------	-------------	------	--------------	---------------------	-----------	------------------	---------------

Luego agregaremos imágenes de nuestros proveedores y así poder visualizarlos al hacer nuevamente select sobre dicha tabla.



```
--HAREMOS UN SELECT SOBRE DICHA TABLA PUDIENDO VERIFICAR QUE LA MISMA--
--SE ENCUENTRA VACIA Y REPETIMOS DICHA ACCION LUEGO DE HABER --
--AGREGADO IMAGENES A LA MISMA--

SELECT * FROM FABRICANTE
GO
```

110 %

Results Messages

	stream_id	file_stream	name
1	093A4664-52B9-ED11-B711-08002752F1D5	0x89504E470D0A1A0A0000000D49484452000000CD000000...	HONDA.png
2	0B3A4664-52B9-ED11-B711-08002752F1D5	0x89504E470D0A1A0A0000000D494844520000009B000000...	HARLEY-DAVIDSON.png
3	0D3A4664-52B9-ED11-B711-08002752F1D5	0x89504E470D0A1A0A0000000D49484452000000AC000000...	YAMAHA.png

## 5.OPERACIONES CON PARTICIONES

[Volver al índice →](#)

En SQL, las particiones se utilizan para dividir grandes conjuntos de datos en partes más pequeñas y manejables, dividiendo sus datos y pudiendo ser distribuidos en filegroups.

Para ello debemos inicialmente crear filegroups que serán quienes contengan las particiones, debe estar acompañada de una función de partición y un esquema de partición.

Aplicándolo a nuestro proyecto, crearemos una tabla de ventas, las cuales serán particionadas en función de la fecha de las mismas, con la finalidad de poder controlar de una manera más rigurosa las mismas y tomar en base a ello decisiones financieras.

Lo primero que haremos será crear un grupo o grupo de archivos y a su vez sus correspondientes particiones.

USE master

Go

-- CREAMOS LA BASE DE DATOS CON SUS ARCHIVOS PRINCIPALES MDF Y LDF

DROP DATABASE IF EXISTS STASHMOTOR\_PARTICIONES

GO

CREATE DATABASE STASHMOTOR\_PARTICIONES

```
ON PRIMARY ( NAME = 'STASHMOTOR_AAPL',
              FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\PARTICIONES\STASHMOTOR_AAPL.mdf' ,
              SIZE = 15360KB , MAXSIZE = UNLIMITED, FILEGROWTH = 0)
LOG ON ( NAME = 'STASHMOTOR_AAPL_log',
        FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\PARTICIONES\STASHMOTOR_AAPL.ldf' ,
        SIZE = 10176KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
```

GO

--CREAMOS LOS FILEGROUP QUE CONTENDRAN LAS PARTICIONES--

ALTER DATABASE STASHMOTOR\_PARTICIONES ADD FILEGROUP

[FG\_ventas\_enero\_marzo]

GO

```
ALTER DATABASE STASHMOTOR_PARTICIONES ADD FILEGROUP  
[FG_ventas_abril_junio]
```

```
GO
```

```
ALTER DATABASE STASHMOTOR_PARTICIONES ADD FILEGROUP  
[FG_ventas_julio_septiembre]
```

```
GO
```

```
ALTER DATABASE STASHMOTOR_PARTICIONES ADD FILEGROUP  
[FG_ventas_octubre_diciembre]
```

```
GO
```

```
--CREAMOS LOS ARCHIVOS--
```

```
ALTER DATABASE STASHMOTOR_PARTICIONES ADD FILE  
( NAME = 'ventas_enero_marzo', FILENAME =  
'C:\Program Files\Microsoft SQL  
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\PARTICIONES\ventas_enero_marzo.ndf'  
,  
SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 2MB )  
TO FILEGROUP [FG_ventas_enero_marzo]  
GO
```

```
ALTER DATABASE STASHMOTOR_PARTICIONES ADD FILE  
( NAME = 'ventas_abril_junio', FILENAME =  
'C:\Program Files\Microsoft SQL  
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\PARTICIONES\ventas_abril_junio.ndf'  
,  
SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 2MB )  
TO FILEGROUP [FG_ventas_abril_junio]  
GO
```

```
ALTER DATABASE STASHMOTOR_PARTICIONES ADD FILE  
( NAME = 'ventas_julio_septiembre', FILENAME =  
'C:\Program Files\Microsoft SQL  
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\PARTICIONES\ventas_julio_septiembr  
e.ndf',  
SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 2MB )  
TO FILEGROUP [FG_ventas_julio_septiembre]  
GO
```

```
ALTER DATABASE STASHMOTOR_PARTICIONES ADD FILE  
( NAME = 'ventas_octubre_diciembre', FILENAME =  
'C:\Program Files\Microsoft SQL  
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\PARTICIONES\ventas_octubre_diciemb  
re.ndf',  
SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 2MB )
```

```

TO FILEGROUP [FG_ventas_octubre_diciembre]
GO
--CREAMOS UNA FUNCION DE PARTICION LA CUAL ASIGNARÁ LOS LIMITES--

--EN PRINCIPIO DIVIDIREMOS EL AÑO EN TRIMESTRES (PRIMEROS 9 MESES)

--ESTO ESTABLECERÁ TRES RANGOS DE FECHAS:
--(TODO LO QUE ESTE ANTES DEL PRIMERO DE MARZO)
--(TODO LO QUE SE ENCUENTRE ENTRE MARZO Y JUNIO)
--(TODO LO QUE ESTE DESPUES DE JUNIO)

CREATE PARTITION FUNCTION FN_venta_fecha (datetime)
AS RANGE RIGHT
    FOR VALUES ('2023-3-1', '2023-6-1')
GO

--CREAMOS UN ESQUEMA DE PARTICION QUE ASIGNARA LAS PARTICIONES--
--DE UNA TABLA O INDICE CON PARTICIONES A LOS NUEVOS GRUPOS DE
ARCHIVOS--

CREATE PARTITION SCHEME venta_fecha
AS PARTITION FN_venta_fecha
    TO
(FG_ventas_enero_marzo,FG_ventas_abril_junio,FG_ventas_julio_septiembre,
FG_ventas_octubre_diciembre)
GO

--CREAMOS LA TABLA VENTAS--

DROP TABLE IF EXISTS VENTA
GO

CREATE TABLE VENTA (
    ID_Venta int IDENTITY(1,1) NOT NULL,
    ID_Cliente int,
    Fecha_Venta datetime,
    Total_Venta money,
)
    ON venta_fecha --ESQUEMA DE PARTICION
    (Fecha_Venta) -- COLUMNA DONDE APLICAREMOS LA FUNCION DENTRO
DEL ESQUEMA
GO

--INSERCIÓN DE DATOS--

```

```
--INGRESAREMOS DATOS Y PODREMOS IR OBSERVANDO COMO VAN SIENDO
DISTRIBUIDOS--
--A LAS DISTINTAS PARTICIONES EN FUNCION DE SU FECHA DE VENTA--
```

```
INSERT INTO VENTA (ID_Cliente, Fecha_Venta, Total_Venta)
VALUES
    (1, '2023-01-17 10:00:00', 100.00),
    (2, '2023-02-12 11:00:00', 200.00),
    (3, '2023-02-25 12:00:00', 150.00),
    (4, '2023-02-28 16:00:00', 300.00)
```

```
GO
```

```
--HACEMOS UN SELECT SOBRE LA TABLA PARA COMPROBAR EN QUE PARTICION SE
ENCUENTRAN LOS REGISTROS--
```

```
SELECT *, $Partition.FN_venta_fecha(fecha_venta) AS Partition
FROM VENTA
GO
```

	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta	Partition
1	1	1	2023-01-17 10:00:00.000	100,00	1
2	2	2	2023-02-12 11:00:00.000	200,00	1
3	3	3	2023-02-25 12:00:00.000	150,00	1
4	4	4	2023-02-28 16:00:00.000	300,00	1

```
--INSERTAMOS NUEVAS VENTAS--
```

```
INSERT INTO VENTA (ID_Cliente, Fecha_Venta, Total_Venta)
VALUES
    (5, '2023-03-17 11:00:00', 75.00),
    (6, '2023-03-12 15:00:00', 20.00),
    (7, '2023-04-25 09:00:00', 180.00),
    (2, '2023-05-28 10:00:00', 370.00)
```

```
GO
```

	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta	Partition
1	1	1	2023-01-17 10:00:00.000	100,00	1
2	2	2	2023-02-12 11:00:00.000	200,00	1
3	3	3	2023-02-25 12:00:00.000	150,00	1
4	4	4	2023-02-28 16:00:00.000	300,00	1
5	5	5	2023-03-17 11:00:00.000	75,00	2
6	6	6	2023-03-12 15:00:00.000	20,00	2
7	7	7	2023-04-25 09:00:00.000	180,00	2
8	8	2	2023-05-28 10:00:00.000	370,00	2

```
INSERT INTO VENTA (ID_Cliente, Fecha_Venta, Total_Venta)
VALUES
```

```
(8, '2023-06-7 11:00:00', 375.00),
(9, '2023-06-12 15:00:00', 220.00),
(10, '2023-07-15 09:00:00', 480.00),
(11, '2023-08-19 10:00:00', 190.00)
```

```
GO
```

	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta	Partition
1	1	1	2023-01-17 10:00:00.000	100,00	1
2	2	2	2023-02-12 11:00:00.000	200,00	1
3	3	3	2023-02-25 12:00:00.000	150,00	1
4	4	4	2023-02-28 16:00:00.000	300,00	1
5	5	5	2023-03-17 11:00:00.000	75,00	2
6	6	6	2023-03-12 15:00:00.000	20,00	2
7	7	7	2023-04-25 09:00:00.000	180,00	2
8	8	2	2023-05-28 10:00:00.000	370,00	2
9	9	8	2023-06-07 11:00:00.000	375,00	3
10	10	9	2023-06-12 15:00:00.000	220,00	3
11	11	10	2023-07-15 09:00:00.000	480,00	3
12	12	11	2023-08-19 10:00:00.000	190,00	3

Podemos observar que en este momento existen 3 filegroup con las 3 particiones, como no hemos establecido particiones adicionales, todos los registros que se han posteriores a la última fecha seguirán formando parte de la partición número 3.

Seguimos insertando datos en nuestra tabla de ventas.

```
INSERT INTO VENTA (ID_Cliente, Fecha_Venta, Total_Venta)
VALUES
```

```
(12, '2023-09-09 11:00:00', 575.00),
(13, '2023-09-13 15:00:00', 1220.00),
(2, '2023-10-16 09:00:00', 780.00),
(14, '2023-11-30 10:00:00', 1190.00)
```

GO

	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta	Partition
1	1	1	2023-01-17 10:00:00.000	100,00	1
2	2	2	2023-02-12 11:00:00.000	200,00	1
3	3	3	2023-02-25 12:00:00.000	150,00	1
4	4	4	2023-02-28 16:00:00.000	300,00	1
5	5	5	2023-03-17 11:00:00.000	75,00	2
6	6	6	2023-03-12 15:00:00.000	20,00	2
7	7	7	2023-04-25 09:00:00.000	180,00	2
8	8	2	2023-05-28 10:00:00.000	370,00	2
9	9	8	2023-06-07 11:00:00.000	375,00	3
10	10	9	2023-06-12 15:00:00.000	220,00	3
11	11	10	2023-07-15 09:00:00.000	480,00	3
12	12	11	2023-08-19 10:00:00.000	190,00	3
13	13	12	2023-09-09 11:00:00.000	575,00	3
14	14	13	2023-09-13 15:00:00.000	1220,00	3
15	15	2	2023-10-16 09:00:00.000	780,00	3
16	16	14	2023-11-30 10:00:00.000	1190,00	3



## 5.1 SPLIT

[Volver al índice →](#)

Establece un límite dentro de la función de partición, dividiendo de esta manera una partición existente en otras más pequeñas, continuando con el ejemplo anterior, nos interesa que las ventas que han sido registradas en el último trimestre del año se ubiquen en una cuarta partición siguiendo con la estructura que se venía trabajando.

Forzando con los ejemplos anteriores que esto sucediera, ya había sido creada la cuarta partición por lo cual solo debemos aplicar los siguientes comandos.

```
--SPLIT--
```

```
ALTER PARTITION FUNCTION FN_venta_fecha()  
    SPLIT RANGE ('2023-09-1');  
GO
```

```
--HACEMOS NUEVAMENTE UN SELECT PARA VERIFICAR LOS CAMBIOS--
```

```
SELECT *, $Partition.FN_venta_fecha(venta_fecha) AS Partition  
FROM VENTA  
GO
```

	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta	Partition
1	1	1	2023-01-17 10:00:00.000	100,00	1
2	2	2	2023-02-12 11:00:00.000	200,00	1
3	3	3	2023-02-25 12:00:00.000	150,00	1
4	4	4	2023-02-28 16:00:00.000	300,00	1
5	5	5	2023-03-17 11:00:00.000	75,00	2
6	6	6	2023-03-12 15:00:00.000	20,00	2
7	7	7	2023-04-25 09:00:00.000	180,00	2
8	8	2	2023-05-28 10:00:00.000	370,00	2
9	9	8	2023-06-07 11:00:00.000	375,00	3
10	10	9	2023-06-12 15:00:00.000	220,00	3
11	11	10	2023-07-15 09:00:00.000	480,00	3
12	12	11	2023-08-19 10:00:00.000	190,00	3
13	13	12	2023-09-09 11:00:00.000	575,00	4
14	14	13	2023-09-13 15:00:00.000	1220,00	4
15	15	2	2023-10-16 09:00:00.000	780,00	4
16	16	14	2023-11-30 10:00:00.000	1190,00	4

## 5.2 MERGE

[Volver al índice →](#)

Nos permite eliminar uno de los límites de una función existente, haciendo que dos o más particiones se fusionen en una sola, continuando con nuestro ejemplo, teniendo en cuenta que hemos segmentado las ventas en trimestres, esta vez nos interesa almacenar en una misma particion todas las ventas realizadas en el primer semestre del año.

Siendo este el caso, eliminaremos el límite existente entre los dos primeros trimestres de la siguiente manera:

```
--MERGE--
```

```
ALTER PARTITION FUNCTION FN_venta_fecha()  
    MERGE RANGE ('2023-3-1');
```

```
GO
```

```
SELECT *,$Partition.FN_venta_fecha(fecha_venta) AS Partition  
FROM VENTA  
GO
```

	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta	Partition
1	1	1	2023-01-17 10:00:00.000	100,00	1
2	2	2	2023-02-12 11:00:00.000	200,00	1
3	3	3	2023-02-25 12:00:00.000	150,00	1
4	4	4	2023-02-28 16:00:00.000	300,00	1
5	5	5	2023-03-17 11:00:00.000	75,00	1
6	6	6	2023-03-12 15:00:00.000	20,00	1
7	7	7	2023-04-25 09:00:00.000	180,00	1
8	8	2	2023-05-28 10:00:00.000	370,00	1
9	9	8	2023-06-07 11:00:00.000	375,00	2
10	10	9	2023-06-12 15:00:00.000	220,00	2
11	11	10	2023-07-15 09:00:00.000	480,00	2
12	12	11	2023-08-19 10:00:00.000	190,00	2
13	13	12	2023-09-09 11:00:00.000	575,00	3
14	14	13	2023-09-13 15:00:00.000	1220,00	3
15	15	2	2023-10-16 09:00:00.000	780,00	3
16	16	14	2023-11-30 10:00:00.000	1190,00	3

## 5.3 SWITCH

[Volver al índice →](#)

Nos permite mover registros de una partición de una tabla previamente particionada, a otra tabla o viceversa (switch in / switch out). Continuando con los casos anteriores, moveremos todas las ventas que se registraron en el primer semestre del año, a una nueva tabla.

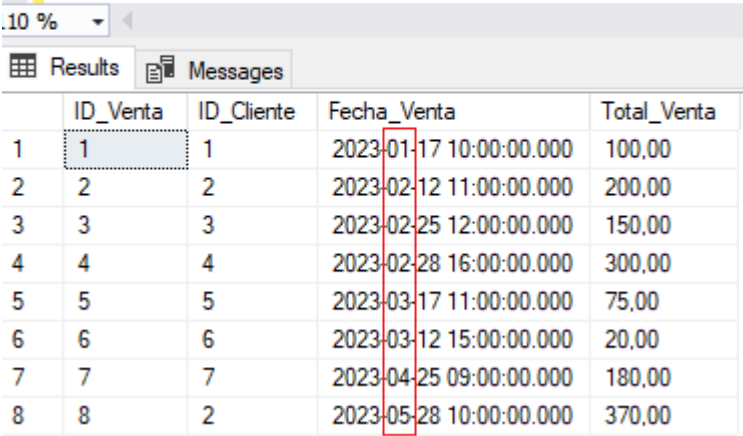
--SWITCH--

```
CREATE TABLE PRIMER_SEMESTRE (
    ID_Venta int IDENTITY(1,1) NOT NULL,
    ID_Cliente int,
    Fecha_Venta datetime,
    Total_Venta money,
)
ON FG_ventas_enero_marzo
go
```

```
ALTER TABLE VENTA
    SWITCH Partition 1 to PRIMER_SEMESTRE
go
```

--HAREMOS UN SELECT DE AMBAS TABLAS PARA VISUALIZAR LOS CAMBIOS--

```
SELECT * FROM PRIMER_SEMESTRE
GO
```

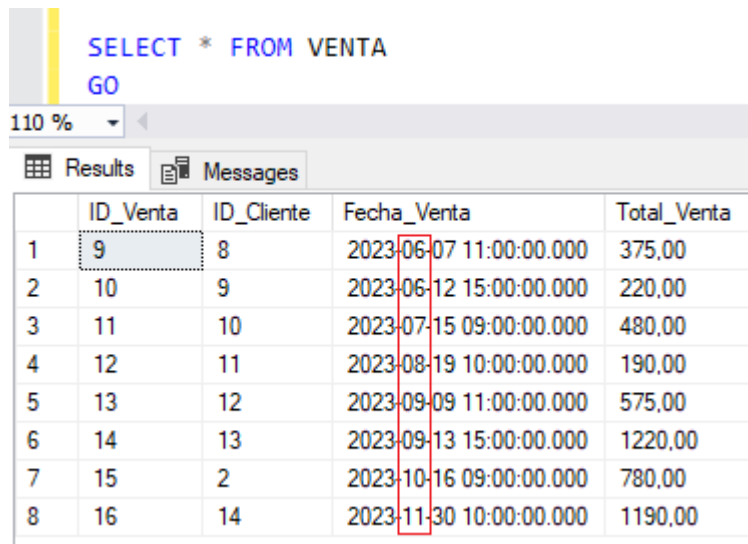


	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta
1	1	1	2023-01-17 10:00:00.000	100,00
2	2	2	2023-02-12 11:00:00.000	200,00
3	3	3	2023-02-25 12:00:00.000	150,00
4	4	4	2023-02-28 16:00:00.000	300,00
5	5	5	2023-03-17 11:00:00.000	75,00
6	6	6	2023-03-12 15:00:00.000	20,00
7	7	7	2023-04-25 09:00:00.000	180,00
8	8	2	2023-05-28 10:00:00.000	370,00

Vemos que efectivamente se han movido las ventas del primer semestre del

año

Al hacer una select en nuestra tabla de venta, confirmamos que solo han quedado los registros pertenecientes a los dos últimos trimestres del año.



The screenshot shows a SQL query execution window. At the top, the query `SELECT * FROM VENTA` is entered, followed by a `GO` command. Below the query bar, a zoom level of 110% is indicated. The interface has two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with 5 columns: an index, `ID_Venta`, `ID_Cliente`, `Fecha_Venta`, and `Total_Venta`. The table contains 8 rows of data. The first row has `ID_Venta` 9, which is highlighted with a dashed border. The `Fecha_Venta` column contains dates from 2023-06-07 to 2023-11-30. A red vertical rectangle highlights the `Fecha_Venta` column, indicating the date range of the data.

	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta
1	9	8	2023-06-07 11:00:00.000	375,00
2	10	9	2023-06-12 15:00:00.000	220,00
3	11	10	2023-07-15 09:00:00.000	480,00
4	12	11	2023-08-19 10:00:00.000	190,00
5	13	12	2023-09-09 11:00:00.000	575,00
6	14	13	2023-09-13 15:00:00.000	1220,00
7	15	2	2023-10-16 09:00:00.000	780,00
8	16	14	2023-11-30 10:00:00.000	1190,00

## 5.4 TRUNCATE

[Volver al índice →](#)

Nos permite eliminar una partición, para demostrarlo, eliminaremos las ventas correspondientes al último trimestre del año, quedando únicamente la partición 2 que comprende los meses de julio a octubre.

```
-- TRUNCATE
```

```
TRUNCATE TABLE VENTA  
    WITH (PARTITIONS (3));  
GO
```

```
SELECT *, $Partition.FN_venta_fecha(venta_fecha) AS Partition  
FROM VENTA  
GO
```

	ID_Venta	ID_Cliente	Fecha_Venta	Total_Venta	Partition
1	9	8	2023-06-07 11:00:00.000	375,00	2
2	10	9	2023-06-12 15:00:00.000	220,00	2
3	11	10	2023-07-15 09:00:00.000	480,00	2
4	12	11	2023-08-19 10:00:00.000	190,00	2

## 6.TABLAS TEMPORALES / VERSIÓN DEL SISTEMA

[Volver al índice →](#)

Es una tabla la cual nos permite almacenar en ella ,como propiamente lo indica, de manera temporal una serie de datos, con la finalidad de realizar consultas con un mejor rendimiento, almacenar datos intermedios, y otras funcionalidades.

Dichas tablas son versión del sistema, ya que su tiempo de validez será administrado por el sistema.

Su forma de implementación es a través de dos tablas: Una tabla actual y una tabla de historial. Dentro de ellas tendremos dos columnas datetime2 para definir el periodo de validez de las mismas.

-Columna de inicio del período: el sistema registra la hora de inicio de la fila en esta columna (SysStartTime).

-Columna de fin del período: El sistema registra la hora de fin de la fila en esta columna (SysEndTime).

En este proyecto aplicaremos dicho ejemplo a la tabla “inventario”, para poder llevar un control del mismo en el tiempo, y los posibles cambios que tenga el mismo.

--TABLAS TEMPORALES / VERSION DEL SISTEMA--

--CREACION BASE DE DATOS--

```
DROP DATABASE IF EXISTS STASHMOTOR_TABLATEMPORAL_VS
GO
```

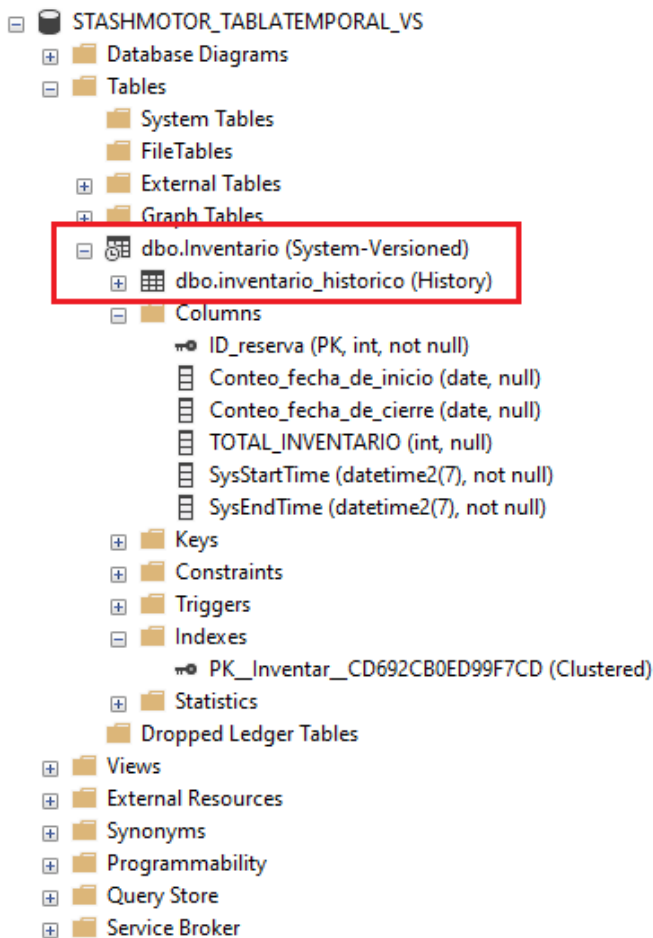
```
CREATE DATABASE STASHMOTOR_TABLATEMPORAL_VS
    ON PRIMARY ( NAME = 'STASHMOTOR_TEMPORAL_VS',
        FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\TEMPORAL_VS\STASHMOTOR_TEMPORAL.MD
F' ,
        SIZE = 15360KB , MAXSIZE = UNLIMITED, FILEGROWTH = 0)
    LOG ON ( NAME = 'STASHMOTOR_TEMPORAL_LOG',
        FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\TEMPORAL_VS\STASHMOTOR_TEMPORAL.LD
F' ,
        SIZE = 10176KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
```

```
USE STASHMOTOR_TABLATEMPORAL_VS
GO
```

```
--CREACIÓN TABLA TEMPORA INVENTARIO--
```

```
CREATE TABLE Inventario
(
    ID_Inventario INTEGER IDENTITY(1,1) NOT NULL Primary Key Clustered,
    Conteo_fecha_de_inicio DATE ,
    Conteo_fecha_de_cierre DATE ,
    TOTAL_INVENTARIO INTEGER ,
    SysStartTime datetime2 generated always as row start not
null,
    SysEndTime datetime2 generated always as row end not null,
    period for System_time (SysStartTime,SysEndTime) )
with (System_Versioning = ON (History_Table =
dbo.inventario_historico))
GO.
```

Comprobamos la correcta creación de ambas tablas desde el entorno gráfico



Ahora procedemos a ingresar datos

```
--INSERCIÓN DE DATOS--

INSERT INTO Inventario (Conteo_fecha_de_inicio,Conteo_fecha_de_cierre,TOTAL_INVENTARIO)
VALUES ('2023-1-01','2023-1-30',25),
       ('2023-2-01','2023-2-28',46),
       ('2023-3-01','2023-3-30',18),
       ('2023-4-01','2023-4-30',52)

GO
```

110 %

Messages

(4 rows affected)

Completion time: 2023-03-06T12:37:38.8451513+01:00

```
SELECT * FROM Inventario
GO
```

110 %

Results Messages

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
2	6	2023-02-01	2023-02-28	46	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
3	7	2023-03-01	2023-03-30	18	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
4	8	2023-04-01	2023-04-30	52	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999

Ahora verificamos nuestra tabla de histórico, como no hemos actualizado datos la misma de momento se encuentra vacía.

```
--VERIFICAMOS LA TABLA DE HISTORICO--

SELECT * FROM Inventario_historico
GO
```

0 %

Results Messages

ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
---------------	------------------------	------------------------	------------------	--------------	------------



Actualizamos la tabla de inventario del mes de abril, comprobamos que se ha modificado el total de inventario.

```

update Inventario
set TOTAL_INVENTARIO = 60
where ID_Inventario = 8
GO

```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
2	6	2023-02-01	2023-02-28	46	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
3	7	2023-03-01	2023-03-30	18	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
4	8	2023-04-01	2023-04-30	60	2023-03-06 11:46:26.1458824	9999-12-31 23:59:59.9999999

Ahora al consultar la tabla de histórico, podremos observar la fecha en que fue modificado el inventario y el valor antiguo del mismo.

```

SELECT * FROM Inventario_historico
GO

```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	8	2023-04-01	2023-04-30	52	2023-03-06 11:37:38.8451513	2023-03-06 11:46:26.1458824

Actualizamos nuevamente la tabla de inventario correspondiente al mes de abril

```

--ACTUALIZAMOS NUEVAMENTE EL INVENTARIO--

update Inventario
set TOTAL_INVENTARIO = 44
where ID_Inventario = 8
GO

```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
2	6	2023-02-01	2023-02-28	46	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
3	7	2023-03-01	2023-03-30	18	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
4	8	2023-04-01	2023-04-30	44	2023-03-06 11:51:38.0052885	9999-12-31 23:59:59.9999999

Consultamos nuevamente el histórico de la tabla de inventario y observamos los cambios.

```
--CONSULTAMOS NUEVAMENTE LA TABLA DE HISTORICO PARA OBSERVAR MAS DETALLADAMENTE LOS CAMBIOS--
```

```
SELECT * FROM Inventario_historico
```

```
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	8	2023-04-01	2023-04-30	52	2023-03-06 11:37:38.8451513	2023-03-06 11:46:26.1458824
2	8	2023-04-01	2023-04-30	60	2023-03-06 11:46:26.1458824	2023-03-06 11:51:38.0052885

Ahora simularemos que fue borrado el inventario correspondiente al mes de enero, pero de igual manera podremos recuperar, ya que se encontrará almacenado en el histórico.

Consultamos la tabla de inventario.

```
--CONSULTAMOS LA TABLA--
```

```
SELECT * FROM Inventario
```

```
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	6	2023-02-01	2023-02-28	46	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
2	7	2023-03-01	2023-03-30	18	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
3	8	2023-04-01	2023-04-30	44	2023-03-06 11:51:38.0052885	9999-12-31 23:59:59.9999999

Una vez eliminado el inventario de enero, consultamos la tabla de histórico y podemos observar que se ha guardado el inventario correspondiente al mes de enero.

```
--CONSULTAMOS LA TABLA HISTORICO--
```

```
SELECT * FROM Inventario_historico
```

```
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	8	2023-04-01	2023-04-30	52	2023-03-06 11:37:38.8451513	2023-03-06 11:46:26.1458824
2	8	2023-04-01	2023-04-30	60	2023-03-06 11:46:26.1458824	2023-03-06 11:51:38.0052885
3	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	2023-03-06 12:01:18.7396093

Insertamos el inventario correspondiente al mes de mayo

```
--INSERTAMOS EL INVENTARIO CORRESPONDIENTE AL MES DE MAYO--
INSERT INTO Inventario (Conteo_fecha_de_inicio,Conteo_fecha_de_cierre,TOTAL_INVENTARIO)
VALUES ('2023-5-01','2023-5-30',78)
GO

--CONSULTAMOS LA TABLA DE INVENTARIO--

SELECT * FROM Inventario
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	6	2023-02-01	2023-02-28	46	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
2	7	2023-03-01	2023-03-30	18	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
3	8	2023-04-01	2023-04-30	44	2023-03-06 11:51:38.0052885	9999-12-31 23:59:59.9999999
4	9	2023-05-01	2023-05-30	78	2023-03-06 12:13:31.6927779	9999-12-31 23:59:59.9999999

No seremos capaces de observar el mismo cambio en la tabla de histórico, ya que aun no ha sufrido un cambio la misma

```
--AL CONSULTAR LA TABLA DE HISTORICO, NO OBSERVAREMOS NINGUN CAMBIO MAS ALLA DE LA NUEVA INSERCIÓN--
--YA QUE AUN NO SE HA REALIZADO NINGUNA MODIFICACIÓN--

SELECT * FROM Inventario_historico
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	8	2023-04-01	2023-04-30	52	2023-03-06 11:37:38.8451513	2023-03-06 11:46:26.1458824
2	8	2023-04-01	2023-04-30	60	2023-03-06 11:46:26.1458824	2023-03-06 11:51:38.0052885
3	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	2023-03-06 12:01:18.7396093

Luego de haber realizado distintos cambios en la tabla de Inventario, podemos utilizar distintas sentencias, las cuales nos permitirán analizar los datos de la misma.

Con “FOR SYSTEM\_TIME ALL” seremos capaces de visualizar todos los cambios que sufrió dicha tabla.

```
--UTILIZANDO "FOR SYSTEM_TIME ALL" OBSERVAMOS--
--TODAS LAS OPERACIONES SOBRE LA TABLA--

SELECT *
FROM Inventario
FOR SYSTEM_TIME ALL
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	6	2023-02-01	2023-02-28	46	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
2	7	2023-03-01	2023-03-30	18	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
3	8	2023-04-01	2023-04-30	44	2023-03-06 11:51:38.0052885	9999-12-31 23:59:59.9999999
4	9	2023-05-01	2023-05-30	78	2023-03-06 12:13:31.6927779	9999-12-31 23:59:59.9999999
5	8	2023-04-01	2023-04-30	52	2023-03-06 11:37:38.8451513	2023-03-06 11:46:26.1458824
6	8	2023-04-01	2023-04-30	60	2023-03-06 11:46:26.1458824	2023-03-06 11:51:38.0052885
7	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	2023-03-06 12:01:18.7396093

Si quisiéramos ver el estado del inventario en un punto determinado podríamos realizar la siguiente consulta

```
--CONSULTAR LA TABLA INVENTARIO EN UN TIEMPO DETERMINADO--
```

```
SELECT *
FROM Inventario
FOR SYSTEM_TIME AS OF '2023-03-06 11:37:38.8451513'
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	6	2023-02-01	2023-02-28	46	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
2	7	2023-03-01	2023-03-30	18	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
3	8	2023-04-01	2023-04-30	52	2023-03-06 11:37:38.8451513	2023-03-06 11:46:26.1458824
4	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	2023-03-06 12:01:18.7396093

También podremos consultar el inventario en un rango de fechas determinado

```
--SI QUISIERAMOS CONSULTAR EL INVENTARIO EN UN RANGO DE FECHA DETERMINADO--
```

```
SELECT *
FROM Inventario
FOR SYSTEM_TIME FROM '2023-03-06 11:51:38.0052885' TO '2023-03-06 12:13:31.6927779'
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	6	2023-02-01	2023-02-28	46	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
2	7	2023-03-01	2023-03-30	18	2023-03-06 11:37:38.8451513	9999-12-31 23:59:59.9999999
3	8	2023-04-01	2023-04-30	44	2023-03-06 11:51:38.0052885	9999-12-31 23:59:59.9999999
4	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	2023-03-06 12:01:18.7396093

Consultarlo sobre un rango de horas determinado

```
--CONSULTARLO EN UN RAGO DE HORAS DETERMINADO--
```

```
SELECT *
FROM Inventario
FOR SYSTEM_TIME CONTAINED IN ('2023-03-06 11:37:38.8451513', '2023-03-06 12:01:18.7396093')
GO
```

	ID_Inventario	Conteo_fecha_de_inicio	Conteo_fecha_de_cierre	TOTAL_INVENTARIO	SysStartTime	SysEndTime
1	8	2023-04-01	2023-04-30	52	2023-03-06 11:37:38.8451513	2023-03-06 11:46:26.1458824
2	8	2023-04-01	2023-04-30	60	2023-03-06 11:46:26.1458824	2023-03-06 11:51:38.0052885
3	5	2023-01-01	2023-01-30	25	2023-03-06 11:37:38.8451513	2023-03-06 12:01:18.7396093

## 7.TABLAS IN MEMORY

[Volver al índice →](#)

Permiten almacenar temporalmente los datos en la memoria principal del servidor en lugar del disco duro. Pueden ser especialmente útiles cuando es necesario un alto rendimiento en la lectura y escritura de dichos datos, puesto que los mismos son almacenados en la memoria RAM.

Sin embargo también pueden tener limitaciones como puede ser espacio en memoria disponible, además los datos almacenados en dichas tablas **no son persistentes** y se pierden si se produce algún fallo o se apaga el servidor.

Crearemos la misma tabla de inventario relacionada a nuestro proyecto pero esta vez optimizada para memoria.

```
--VERIFICAMOS LA EXISTENCIA DE LA BASE DE DATOS--
```

```
DROP DATABASE IF EXISTS STASHMOTOR_TABLAINMEMORY
GO
```

```
CREATE DATABASE STASHMOTOR_TABLAINMEMORY
go
```

```
USE STASHMOTOR_TABLAINMEMORY
go
```

```
ALTER DATABASE CURRENT
    SET MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT = ON;
GO
```

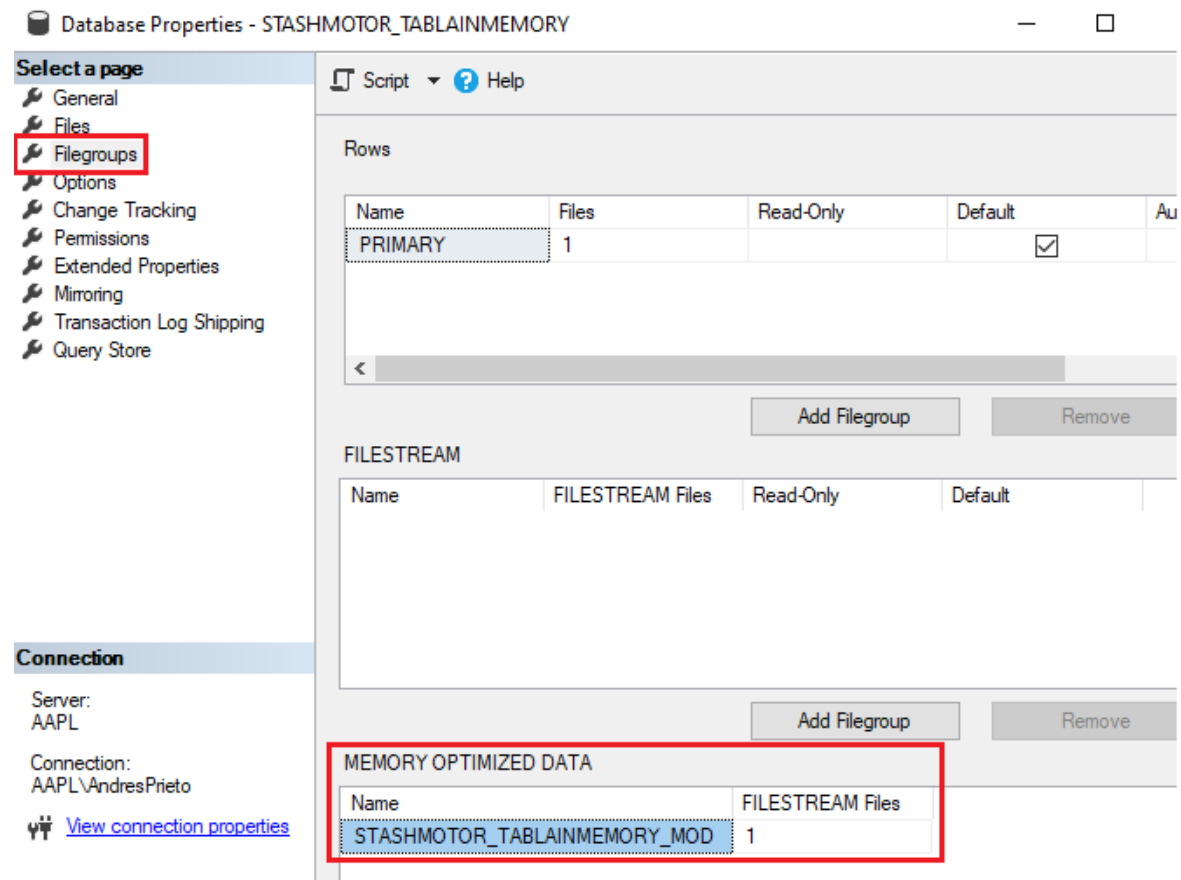
```
--CREAMOS EL FILEGROUP OPTIMIZADO--
```

```
ALTER DATABASE STASHMOTOR_TABLAINMEMORY
    ADD FILEGROUP STASHMOTOR_TABLAINMEMORY_MOD
    CONTAINS MEMORY_OPTIMIZED_DATA
GO
```

```
--AÑADIMOS ARCHIVOS AL MEMORY_OPTIMIZED_DATA FILEGROUP--
```

```
ALTER DATABASE STASHMOTOR_TABLAINMEMORY
    ADD FILE (name='STASHMOTOR_TABLAINMEMORY_MOD1',
    filename='C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\INMEMORY\STASHMOTOR_TABLAINMEMORY_
```

```
MOD1')
    TO FILEGROUP STASHMOTOR_TABLAINMEMORY_MOD
go
```



```
-- Creación de la tabla optimizada para memoria
DROP TABLE IF EXISTS Inventario
GO

CREATE TABLE Inventario(
    [ID_Inventario] [int] IDENTITY(1,1) NOT NULL,
    [Conteo_fecha_de_inicio] [date] NULL,
    [Conteo_fecha_de_cierre] [date] NULL,
    [TOTAL_INVENTARIO] [int] NULL,
    CONSTRAINT [Inventario_PK] PRIMARY KEY NONCLUSTERED (
        [ID_Inventario] ASC
    )
)
WITH
    (MEMORY_OPTIMIZED = ON,
    DURABILITY = SCHEMA_AND_DATA)
GO
```

Table Properties - Inventario

Select a page

- General
- Permissions
- Change Tracking
- Storage
- Security Predicates

Connection

Server:  
AAPL

Connection:  
AAPL\AndresPrieto

Script Help

Current connection parameters

User	AAPL\AndresPrieto
Server	AAPL
Database	STASHMOTOR_TABLAINMEMORY

Description

System object	False
Created date	06/03/2023 15:04
Name	Inventario
Schema	dbo

Options

Quoted identifier	True
ANSI NULLs	True
Memory optimized	True
Durability	SchemaAndData
Table temporal type	None

Replication

Table is replicated	False
---------------------	-------