

**UNIVERSIDAD PRIVADA DE TACNA  
FACULTAD DE INGENIERÍA  
ESCUELA DE INGENIERÍA DE SISTEMAS**



# **LABORATORIO 03**

## **“AZURE”**

**Que se presenta para el curso:**

**“BASE DE DATOS II”**

**Docente:**

Mtro. Ing. JUAN MIGUEL CHOQUE FLORES

**Estudiante:**

Anampa Pancca, David Jordan

**TACNA – PERÚ  
2025**



# Índice General

Índice General	2
Introducción	3
1. Información	4
1.1. Objetivos	4
1.2. Materiales	4
2. DESARROLLO	5
3. Conclusiones	34
4. Referencias Bibliográficas	34

# Introducción

En la actualidad, la administración eficiente de bases de datos y el monitoreo constante de los servicios que interactúan con ellas son elementos fundamentales en entornos empresariales y de desarrollo. Herramientas como **Azure Data Studio** proporcionan un entorno moderno y versátil para la gestión, consulta y visualización de bases de datos, facilitando tareas tanto para administradores como para desarrolladores.

Este proceso no solo permite verificar la correcta funcionalidad de la base de datos, sino también establecer un entorno de observabilidad que facilite la detección de cuellos de botella y el análisis del rendimiento. La integración de todos los componentes se realizará mediante **contenedores Docker**, lo que permitirá una ejecución automatizada, reproducible y portátil del entorno de pruebas.

# **LABORATORIO 03 “Azure”**

## **1. Información**

### **1.1. Objetivos:**

- Restaurar una base de datos en **Azure SQL Database** a partir de un archivo de respaldo.
- Ejecutar consultas SQL básicas sobre la base de datos restaurada.

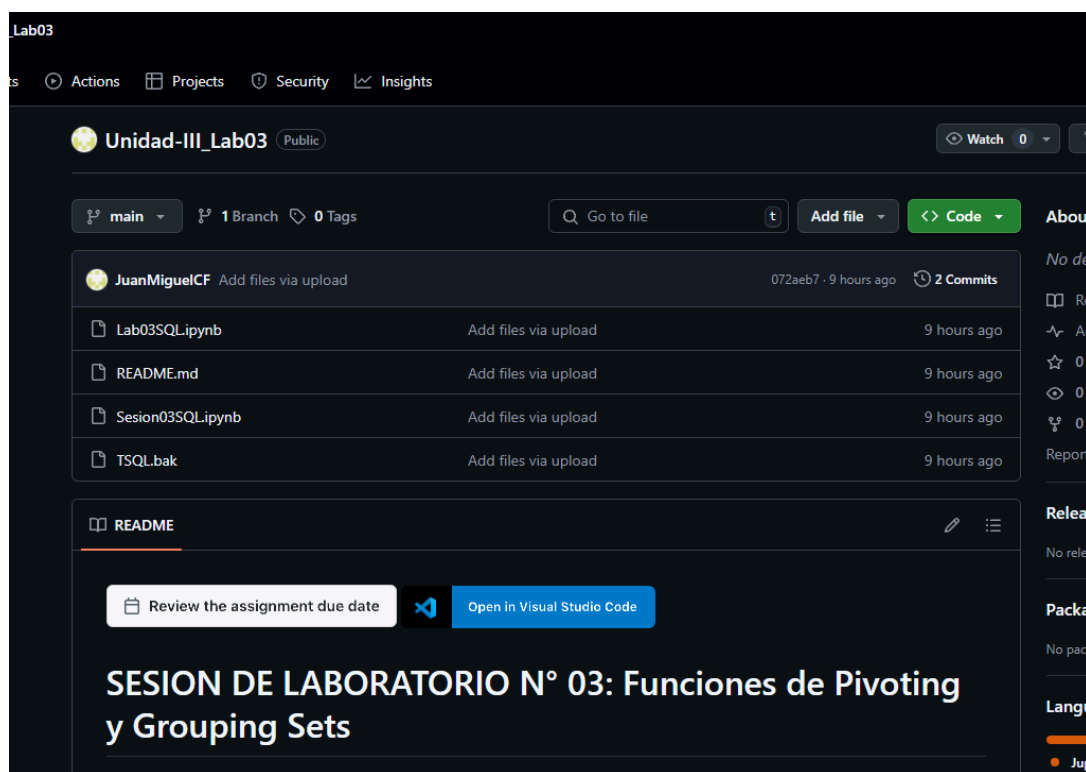
### **1.2. Materiales**

- Visual Studio Code (Editor de código)
- .NET SDK 7.0 o superior (Para desarrollar la API)
- Navegador web (Para visualizar el dashboard de Grafana)
- API en .NET (Proyecto que interactúe con SQL Server)
- Azure Portal (Para gestionar SQL Database y almacenamiento)
- Docker + Docker Compose

## 2. Desarrollo

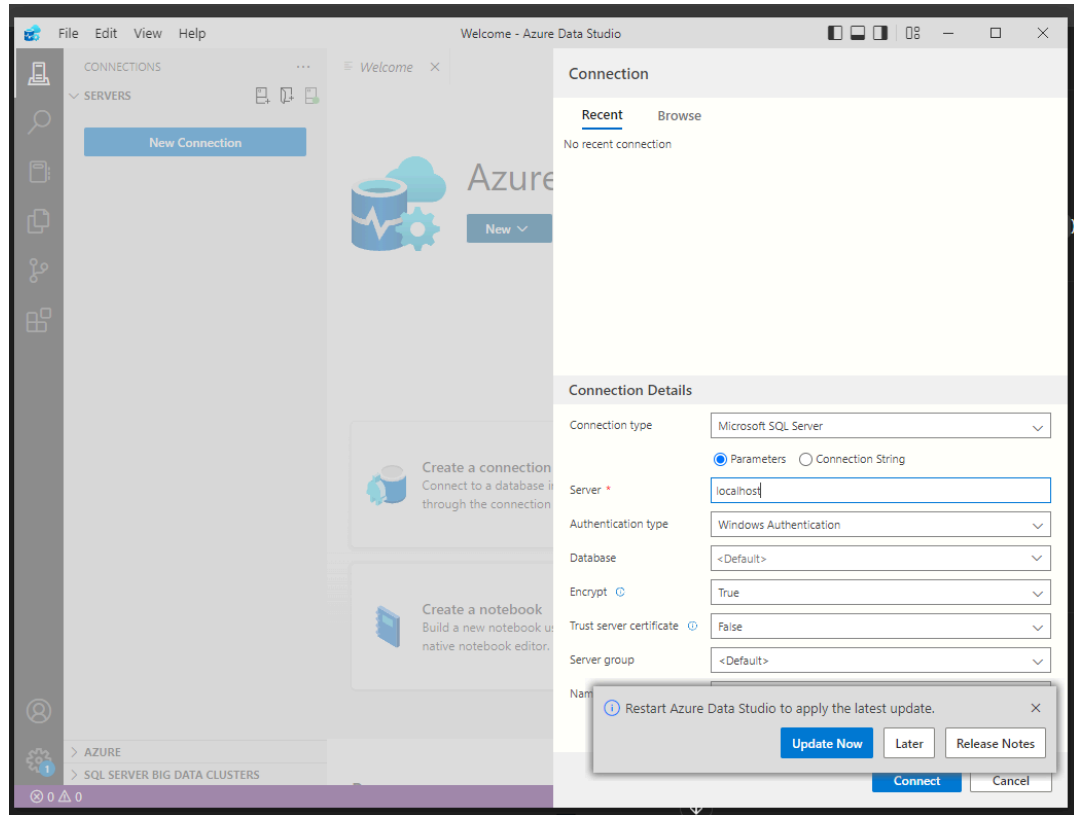
### 1. RESTAURACIÓN DE BASE DE DATOS TSQL.BAK

Primero, ingresamos al enlace de GitHub proporcionado para el laboratorio. Una vez dentro, ubicamos el enlace del repositorio que contiene todos los archivos necesarios para la práctica.

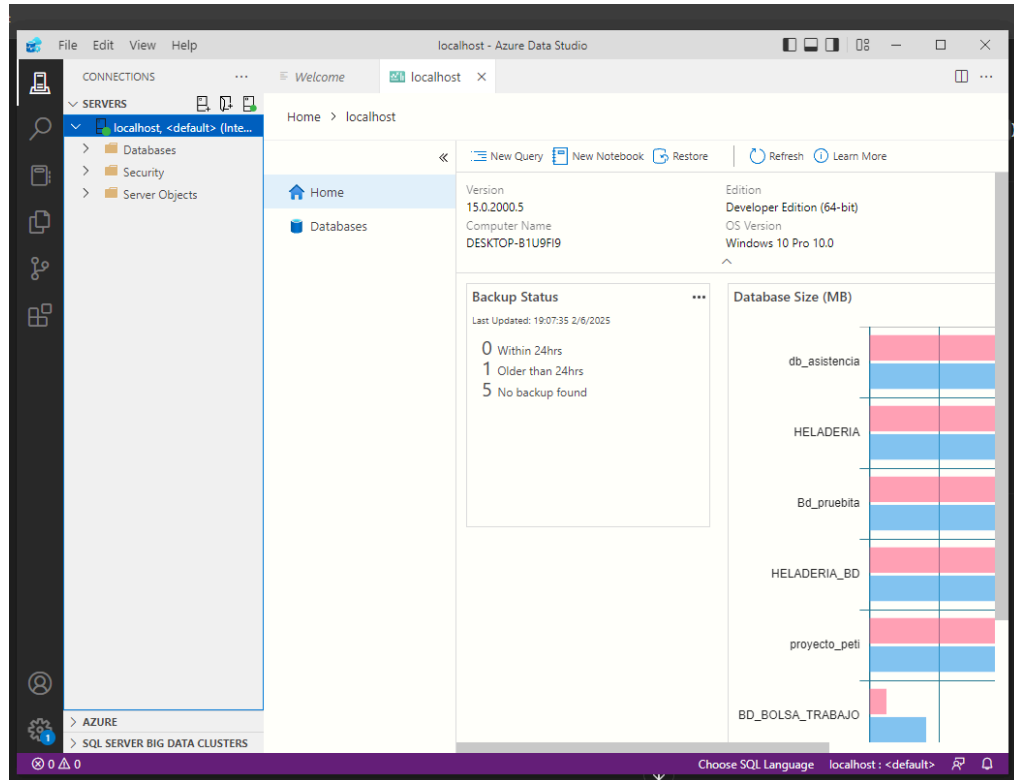


Luego, abrimos Git Bash, una terminal o PowerShell, para clonar el repositorio alojado en GitHub en la ubicación local donde deseamos trabajar. Utilizamos el comando `git clone` seguido de la URL del repositorio para descargar todos los archivos del proyecto.

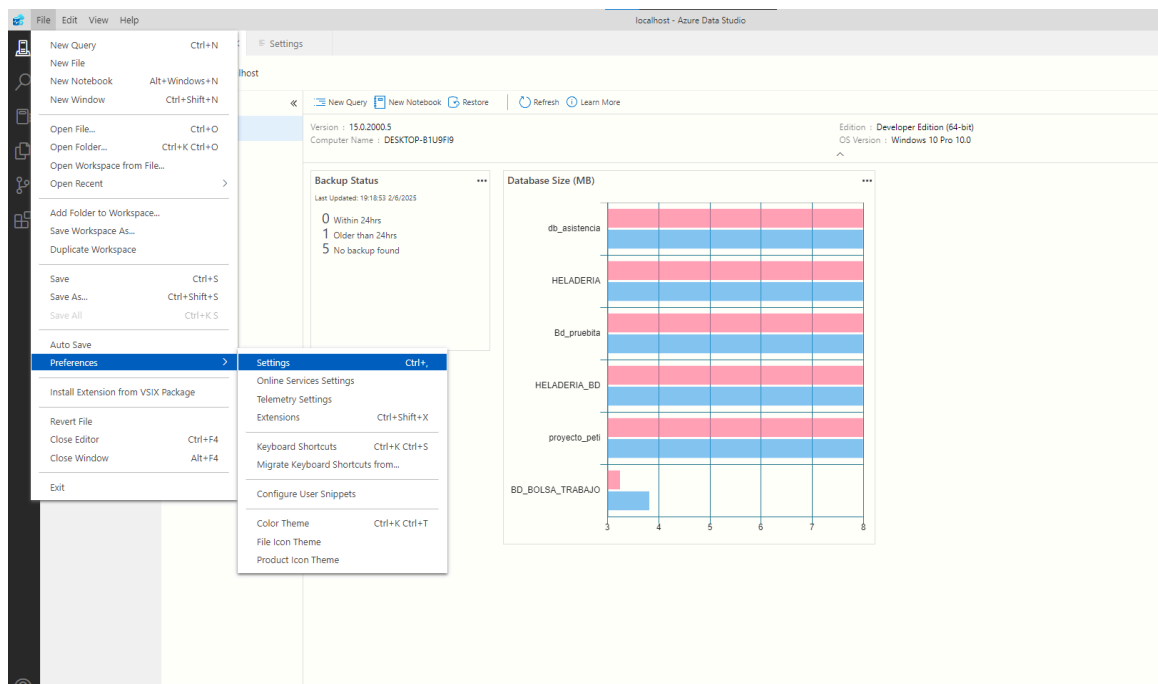
Una vez clonado el repositorio, abrimos Azure Data Studio e iniciamos la conexión al servidor local utilizando el usuario `localhost`



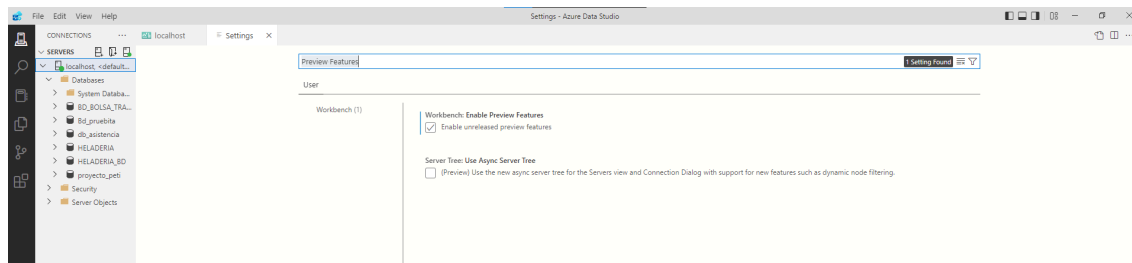
En el apartado de inicio de sesión de Azure Data Studio, si seleccionamos la opción Restore, no podremos restaurar la base de datos directamente, ya que aparecerá un mensaje indicando que la opción de restauración no está habilitada. Esto sucede porque Azure Data Studio, por defecto, no permite restaurar archivos .bak en instancias de SQL Server que estén corriendo en contenedores sin la configuración adecuada.



Para ello nos dirigimos a File>Preferences>Settings, luego nos dirigimos al buscador de settings.



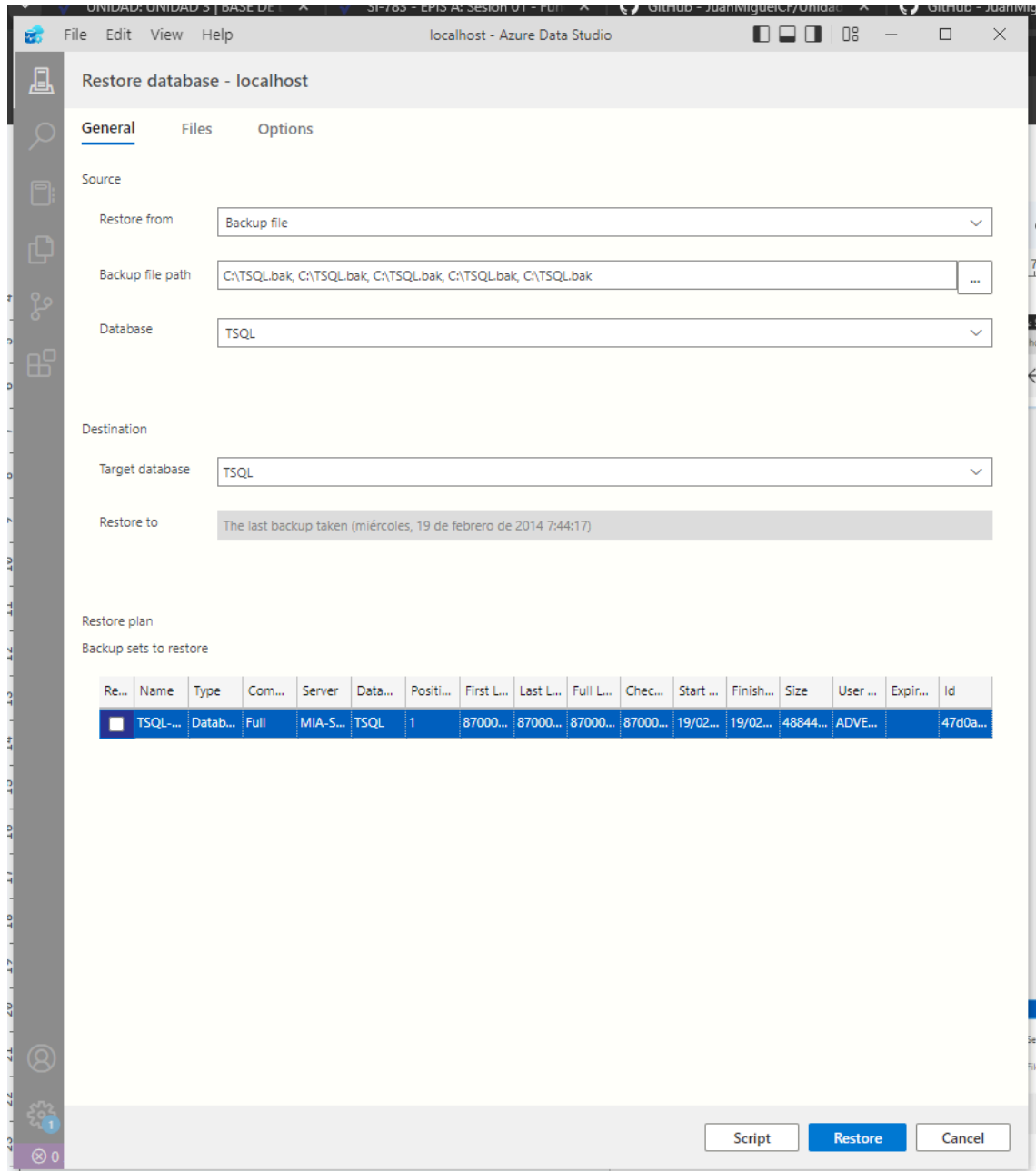
En el buscador, ingresamos el Preview Features y habilitamos la opción del recuadro para poder restaurar nuestra base de datos.



Nuevamente nos dirigimos a la pestaña principal y seleccionamos la opción Restore, esto abrirá una pestaña para seleccionar la ubicación de la base de datos, una vez cargado la ubicación el TSQL.bak, automáticamente se cargará el nombre de la base de datos, etc.



Una vez visualizado el nombre, columnas, usuarios, etc de la Base de Datos TSQL, seleccionamos en la opción Restore para restaurar la base de datos. Después de la restauración visualizamos las tablas que tiene la base de datos TSQL



## **Parte 01. Escribir consultas que utilizan PIVOT.**

### **1.1. Abrir la base de datos TSQL.**

#### **Explicación:**

Este paso indica que vamos a utilizar la base de datos llamada TSQL para ejecutar consultas dentro de ella. En SQL Server, el comando USE se emplea para cambiar el contexto de la base de datos activa. Esto es importante porque, si no se especifica una base de datos, las consultas podrían ejecutarse en otra base por defecto.

#### **Código:**

```
USE TSQL;  
GO
```

#### **Resultado:**



Los comandos se han completado correctamente.

Tiempo total de ejecución: 00:00:00.001

### **1.2. Crear un vista de categorías de productos, sus cantidades por año.**

#### **Explicación:**

En este paso crearemos una vista llamada Sales.CategoryQtyYear, la cual nos permitirá ver las categorías de productos, la cantidad vendida y el año en que se realizaron los pedidos.

Primero, verificamos si la vista ya existe usando OBJECT\_ID, y si es así, la eliminamos para evitar errores al volver a crearla. Luego, usamos CREATE VIEW para definir la vista con una consulta que une varias tablas relacionadas: Categorías, Productos, Detalles de pedidos y Pedidos.

Esta vista nos servirá para consultar fácilmente cuántos productos de cada categoría se vendieron en cada año.

**Código:**

```
IF OBJECT_ID('Sales.CategoryQtyYear','V') IS NOT NULL DROP VIEW
Sales.CategoryQtyYear
GO
CREATE VIEW Sales.CategoryQtyYear
AS
SELECT c.categoryname AS Category,
       od.qty AS Qty,
       YEAR(o.orderdate) AS Orderyear
FROM Production.Categories AS c
      INNER JOIN Production.Products AS p ON c.categoryid=p.categoryid
      INNER JOIN Sales.OrderDetails AS od ON p.productid=od.productid
      INNER JOIN Sales.Orders AS o ON od.orderid=o.orderid;
GO
```

**Resultado:**

```
1.2. Crear un vista de categorías de productos, sus cantidades por año.

[3] 1 IF OBJECT_ID('Sales.CategoryQtyYear','V') IS NOT NULL DROP VIEW Sales.CategoryQtyYear
      2 GO
      3 CREATE VIEW Sales.CategoryQtyYear
      4 AS
      5 SELECT c.categoryname AS Category,
      6        od.qty AS Qty,
      7        YEAR(o.orderdate) AS Orderyear
      8 FROM Production.Categories AS c
      9        INNER JOIN Production.Products AS p ON c.categoryid=p.categoryid
     10        INNER JOIN Sales.OrderDetails AS od ON p.productid=od.productid
     11        INNER JOIN Sales.Orders AS o ON od.orderid=o.orderid;
     12 GO

Los comandos se han completado correctamente.

Los comandos se han completado correctamente.

Tiempo total de ejecución: 00:00:00.017
```

### 1.3. Probar la vista creada.

#### Explicación:

En este paso vamos a probar si la vista Sales.CategoryQtyYear se creó correctamente y devuelve los datos esperados. Para eso, realizamos una consulta SELECT sobre la vista, solicitando que nos muestre las columnas: Category (nombre de la categoría), Qty (cantidad vendida) y Orderyear (año del pedido).

#### Código:

```
SELECT Category, Qty, Orderyear FROM Sales.CategoryQtyYear;
```

#### Resultado:

(2155 filas afectadas)

Tiempo total de ejecución: 00:00:00.060



	Category ▾	Qty ▾	Orderyear ▾
1	Dairy Products	12	2006
2	Grains/Cereals	10	2006
3	Dairy Products	5	2006
4	Produce	9	2006
5	Produce	40	2006
6	Seafood	10	2006
7	Produce	35	2006
8	Condiments	15	2006
9	Grains/Cereals	6	2006
10	Grains/Cereals	15	2006
11	Condiments	20	2006
12	Confections	40	2006
13	Dairy Products	25	2006
14	Dairy Products	40	2006
15	Dairy Products	20	2006
16	Beverages	42	2006
17	Confections	40	2006
18	Beverages	15	2006
19	Meat/Poultry	21	2006
20	Produce	21	2006

#### 1.4. Utilizar PIVOT entre categorías y años de ordenes:

##### Explicación:

En este paso vamos a transformar los datos de la vista Sales.CategoryQtyYear usando la cláusula PIVOT. Esto nos permite reorganizar los datos para que las cantidades (Qty) se distribuyan por año en columnas separadas (2006, 2007, 2008), lo que facilita la comparación de ventas por categoría y por año.

Primero, seleccionamos los datos desde la vista, y luego aplicamos el PIVOT para sumar las cantidades por cada año. Finalmente, ordenamos el resultado por categoría.

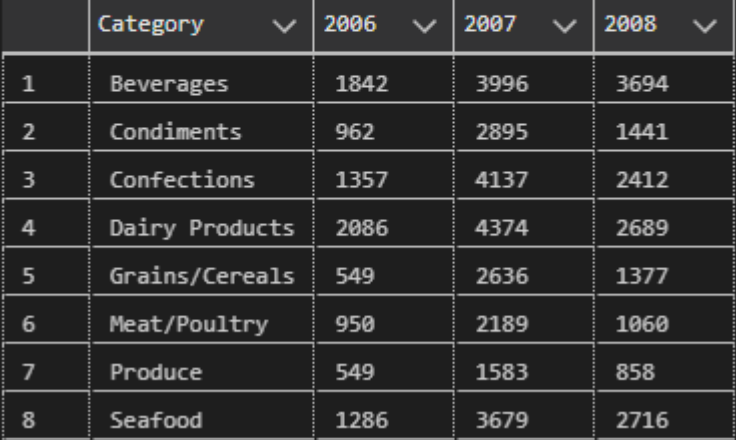
##### Código:

```
SELECT Category, [2006],[2007],[2008]
FROM      ( SELECT      Category, Qty, Orderyear FROM
Sales.CategoryQtyYear) AS D
      PIVOT(SUM(QTY) FOR orderyear IN ([2006],[2007],[2008])) AS
pvt
ORDER BY Category;
```

##### Resultado:

(8 filas afectadas)

Tiempo total de ejecución: 00:00:00.027



	Category	2006	2007	2008
1	Beverages	1842	3996	3694
2	Condiments	962	2895	1441
3	Confections	1357	4137	2412
4	Dairy Products	2086	4374	2689
5	Grains/Cereals	549	2636	1377
6	Meat/Poultry	950	2189	1060
7	Produce	549	1583	858
8	Seafood	1286	3679	2716

### 1.5. Para utilizar UNPIVOT, crear una tabla con el resultado de la consulta con PIVOT anterior:

#### Explicación:

En este paso vamos a crear una tabla llamada Sales.PivotedCategorySales, donde almacenaremos el resultado de la consulta con PIVOT que hicimos anteriormente. Esto es necesario porque el operador UNPIVOT no se puede aplicar directamente sobre una vista, así que primero guardamos los datos pivotados en una tabla real.

Primero usamos CREATE TABLE para definir la estructura de la nueva tabla con las columnas Category, 2006, 2007 y 2008. Luego usamos INSERT INTO con una consulta PIVOT para llenar la tabla con los datos desde la vista Sales.CategoryQtyYear.

#### Código:

```
CREATE TABLE [Sales].[PivotedCategorySales](
    [Category] [nvarchar](15) NOT NULL,
    [2006] [int] NULL,
    [2007] [int] NULL,
    [2008] [int] NULL);
GO
INSERT INTO Sales.PivotedCategorySales (Category,
[2006],[2007],[2008])
SELECT Category, [2006],[2007],[2008]
FROM (SELECT Category, Qty, Orderyear FROM
Sales.CategoryQtyYear) AS D
    PIVOT(SUM(QTY) FOR orderyear IN ([2006],[2007],[2008]))AS p
GO
```

**Resultado:**

1.5. Para utilizar UNPIVOT, crear una tabla con el resultado de la consulta con PIVOT anterior.

```
1 CREATE TABLE [Sales].[PivotedCategorySales](
2     [Category] [nvarchar](15) NOT NULL,
3     [2006] [int] NULL,
4     [2007] [int] NULL,
5     [2008] [int] NULL;
6 GO
7 INSERT INTO Sales.PivotedCategorySales (Category, [2006],[2007],[2008])
8 SELECT Category, [2006],[2007],[2008]
9 FROM (SELECT Category, Qty, Orderyear FROM Sales.CategoryQtyYear) AS D
10     PIVOT(SUM(QTY) FOR orderyear IN ([2006],[2007],[2008])) AS p
11 GO
```

Los comandos se han completado correctamente.

(8 filas afectadas)

Tiempo total de ejecución: 00:00:00.039

**1.6. Probar la tabla generada:****Explicación:**

En este paso vamos a verificar que la tabla Sales.PivotedCategorySales se haya creado correctamente y que contenga los datos esperados. Para eso, realizamos una consulta SELECT que muestra las columnas Category, 2006, 2007 y 2008.

Este paso nos ayuda a confirmar que el resultado del PIVOT fue insertado correctamente en la nueva tabla, y que los datos están listos para ser usados en el siguiente paso con UNPIVOT.

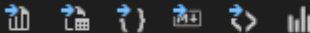
**Código:**

```
SELECT Category, [2006],[2007],[2008]
FROM Sales.PivotedCategorySales;
```

**Resultado:**

(8 filas afectadas)

Tiempo total de ejecución: 00:00:00.007



	Category ▾	2006 ▾	2007 ▾	2008 ▾
1	Beverages	1842	3996	3694
2	Condiments	962	2895	1441
3	Confections	1357	4137	2412
4	Dairy Products	2086	4374	2689
5	Grains/Cereals	549	2636	1377
6	Meat/Poultry	950	2189	1060
7	Produce	549	1583	858
8	Seafood	1286	3679	2716

**1.7. Utilizar UNPIVOT:****Explicación:**

En este paso vamos a aplicar el operador UNPIVOT sobre la tabla Sales.PivotedCategorySales. El objetivo es transformar las columnas de los años 2006, 2007 y 2008 nuevamente en filas, obteniendo así un formato más tradicional de tabla con tres columnas: Category, Qty (cantidad) y Orderyear (año del pedido).

**Código:**

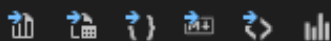
```
SELECT category, qty, orderyear
FROM Sales.PivotedCategorySales
UNPIVOT(qty FOR orderyear IN([2006],[2007],[2008])) AS unpvt;
```



### Resultado:

(24 filas afectadas)

Tiempo total de ejecución: 00:00:00.029



	category	qty	orderyear
1	Beverages	1842	2006
2	Beverages	3996	2007
3	Beverages	3694	2008
4	Condiments	962	2006
5	Condiments	2895	2007
6	Condiments	1441	2008
7	Confections	1357	2006
8	Confections	4137	2007
9	Confections	2412	2008
10	Dairy Products	2086	2006
11	Dairy Products	4374	2007
12	Dairy Products	2689	2008
13	Grains/Cereals	549	2006
14	Grains/Cereals	2636	2007
15	Grains/Cereals	1377	2008
16	Meat/Poultry	950	2006
17	Meat/Poultry	2189	2007
18	Meat/Poultry	1060	2008
19	Produce	549	2006
20	Produce	1583	2007

## 1.8. Limpiar los cambios realizados:

### Explicación:

En este último paso vamos a eliminar la vista y la tabla que creamos durante el laboratorio. Esto es una buena práctica cuando trabajamos con objetos temporales o de prueba, ya que mantiene la base de datos limpia y organizada.

Utilizamos OBJECT\_ID para verificar si la vista Sales.CategoryQtyYear y la tabla Sales.PivotedCategorySales existen. Si existen, se eliminan con DROP VIEW y DROP TABLE respectivamente.

### Código:

```
IF OBJECT_ID('Sales.CategoryQtyYear','V') IS NOT NULL DROP  
VIEW Sales.CategoryQtyYear  
IF OBJECT_ID('Sales.PivotedCategorySales') IS NOT NULL DROP  
TABLE Sales.PivotedCategorySales  
GO
```

### Resultado:

1.8. Limpiar los cambios realizados.

```
[ 9]  1  IF OBJECT_ID('Sales.CategoryQtyYear','V') IS NOT NULL DROP VIEW Sales.CategoryQtyYear  
      2  IF OBJECT_ID('Sales.PivotedCategorySales') IS NOT NULL DROP TABLE Sales.PivotedCategorySales  
      3  GO
```

Los comandos se han completado correctamente.

Tiempo total de ejecución: 00:00:00.014

## Parte 02. Escribir consultas que utilizan UNPIVOT.

### 2.1. Ejecutar la siguiente vistas de:

#### Explicación:

En este paso vamos a crear una nueva vista llamada Sales.CategorySales. Esta vista mostrará información detallada de las ventas de productos, incluyendo la categoría del producto, el ID del empleado que gestionó la venta, el ID del cliente, la cantidad vendida y el año del pedido.

La consulta se enfoca solo en las categorías 1, 2 y 3, y en los clientes con ID del 1 al 5, gracias al uso de la cláusula WHERE. Esta vista nos permitirá analizar las ventas específicas de esas categorías y clientes seleccionados.

Antes de crearla, se verifica si la vista ya existe con OBJECT\_ID, y si es así, se elimina con DROP VIEW para evitar errores al volver a crearla.

#### Código:

```
IF OBJECT_ID('Sales.CategorySales','V') IS NOT NULL DROP VIEW
Sales.CategorySales
GO
CREATE VIEW Sales.CategorySales
AS
SELECT c.categoryname AS Category,
       o.empid AS Emp,
       o.custid AS Cust,
       od.qty AS Qty,
       YEAR(o.orderdate) AS Orderyear
FROM   Production.Categories AS c
       INNER JOIN Production.Products AS p ON c.categoryid=p.categoryid
       INNER JOIN Sales.OrderDetails AS od ON p.productid=od.productid
       INNER JOIN Sales.Orders AS o ON od.orderid=o.orderid
WHERE  c.categoryid IN (1,2,3) AND o.custid BETWEEN 1 AND 5;
GO
```

#### Resultado:

Los comandos se han completado correctamente.

Los comandos se han completado correctamente.

Tiempo total de ejecución: 00:00:00.007

## 2.2. Elaborar una consulta sin utilizar GROUPING SETS:

### Explicación:

En este paso vamos a realizar una consulta que nos dé un resumen de las ventas totales por categoría, por cliente y un total general, pero sin usar GROUPING SETS.

Para lograr esto, usamos la cláusula UNION ALL para combinar tres consultas SELECT:

1. La primera se agrupa por Category, mostrando el total de cantidades por cada categoría.
2. La segunda se agrupa por Cust (cliente), mostrando el total de cantidades por cada cliente.
3. La tercera muestra el total general de todas las cantidades, sin agrupar por ninguna columna.

En cada parte, se usa NULL para las columnas que no se están agrupando, lo que nos permite combinar los resultados en una sola tabla de salida.

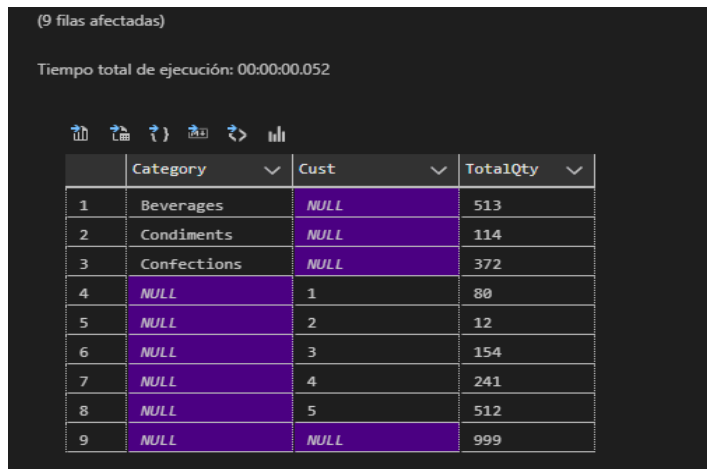
### Código:

```
SELECT Category, NULL AS Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY category
UNION ALL
SELECT NULL, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY cust
UNION ALL
SELECT NULL, NULL, SUM(Qty) AS TotalQty
FROM Sales.CategorySales;
```

### Resultado:

(9 filas afectadas)

Tiempo total de ejecución: 00:00:00.052



	Category	Cust	TotalQty
1	Beverages	NULL	513
2	Condiments	NULL	114
3	Confections	NULL	372
4	NULL	1	80
5	NULL	2	12
6	NULL	3	154
7	NULL	4	241
8	NULL	5	512
9	NULL	NULL	999

### 2.3. Consultar con GROUPING SETS:

#### Explicación:

En este paso vamos a realizar la misma consulta resumen del paso anterior, pero esta vez usando la cláusula GROUPING SETS, que es una forma más eficiente y clara de obtener distintos niveles de agregación en una sola consulta GROUP BY.

Con GROUPING SETS, especificamos los grupos que queremos:

- (Category) para total por categoría.
- (Cust) para total por cliente.
- () para obtener el total general.

Esto simplifica el código, ya que no necesitamos hacer varias consultas con UNION ALL.

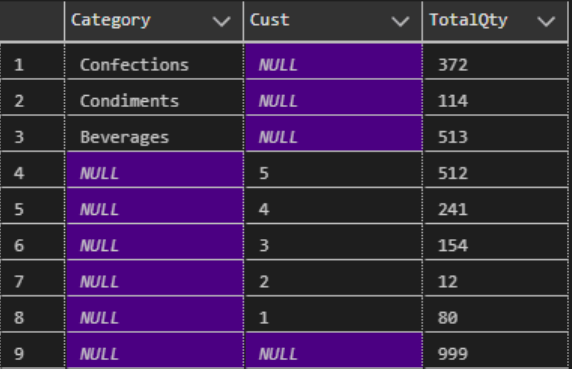
#### Código:

```
SELECT Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY
GROUPING SETS((Category),(Cust),())
ORDER BY Category DESC, Cust desc;
```

#### Resultado:

(9 filas afectadas)

Tiempo total de ejecución: 00:00:00.029



	Category	Cust	TotalQty
1	Confections	NULL	372
2	Condiments	NULL	114
3	Beverages	NULL	513
4	NULL	5	512
5	NULL	4	241
6	NULL	3	154
7	NULL	2	12
8	NULL	1	80
9	NULL	NULL	999

## 2.4. Consultar con CUBE:

### Explicación:

En este paso vamos a usar la cláusula CUBE dentro de GROUP BY, que nos permite generar todas las combinaciones posibles de agregación entre las columnas Category y Cust (cliente).

Al usar GROUP BY CUBE(Category, Cust), obtenemos automáticamente:

- Totales por Category,
- Totales por Cust,
- El total general,
- Y además, el total por cada combinación de Category y Cust.

CUBE es una forma compacta y poderosa de generar múltiples niveles de resumen en una sola consulta.

### Código:

```
SELECT Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY CUBE(Category,Cust)
ORDER BY Category, Cust;
```

### Resultado:

(21 filas afectadas)

Tiempo total de ejecución: 00:00:00.018

	Category	Cust	TotalQty
1	NULL	NULL	999
2	NULL	1	80
3	NULL	2	12
4	NULL	3	154
5	NULL	4	241
6	NULL	5	512
7	Beverages	NULL	513
8	Beverages	1	36
9	Beverages	2	5
10	Beverages	3	105
11	Beverages	4	112
12	Beverages	5	255
13	Condiments	NULL	114
14	Condiments	1	44
15	Condiments	3	4
16	Condiments	5	66
17	Confections	NULL	372
18	Confections	2	7
19	Confections	3	45
20	Confections	4	129

## 2.5. Consultar con ROLLUP:

### Explicación:

En este paso vamos a utilizar la cláusula ROLLUP para agrupar los datos y obtener totales jerárquicos. A diferencia de CUBE, que muestra todas las combinaciones posibles entre las columnas, ROLLUP sigue un orden jerárquico: agrupa primero por Category y luego por Cust, generando subtotales progresivos y un total general.

Con ROLLUP(Category, Cust), primero se muestran los totales por cada combinación de categoría y cliente. Luego, se muestra un subtotal por cada categoría (agrupando todos sus clientes). Finalmente, al final de la consulta, se muestra el total general de todas las cantidades.

### Código:

```
SELECT Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY ROLLUP(Category,Cust)
ORDER BY Category, Cust;
```

### Resultado:

(16 filas afectadas)

Tiempo total de ejecución: 00:00:00.025

	Category	Cust	TotalQty
1	NULL	NULL	999
2	Beverages	NULL	513
3	Beverages	1	36
4	Beverages	2	5
5	Beverages	3	105
6	Beverages	4	112
7	Beverages	5	255
8	Condiments	NULL	114
9	Condiments	1	44
10	Condiments	3	4
11	Condiments	5	66
12	Confections	NULL	372
13	Confections	2	7
14	Confections	3	45
15	Confections	4	129
16	Confections	5	191

## 2.6. Utilizar Grouping\_ID:

### Explicación:

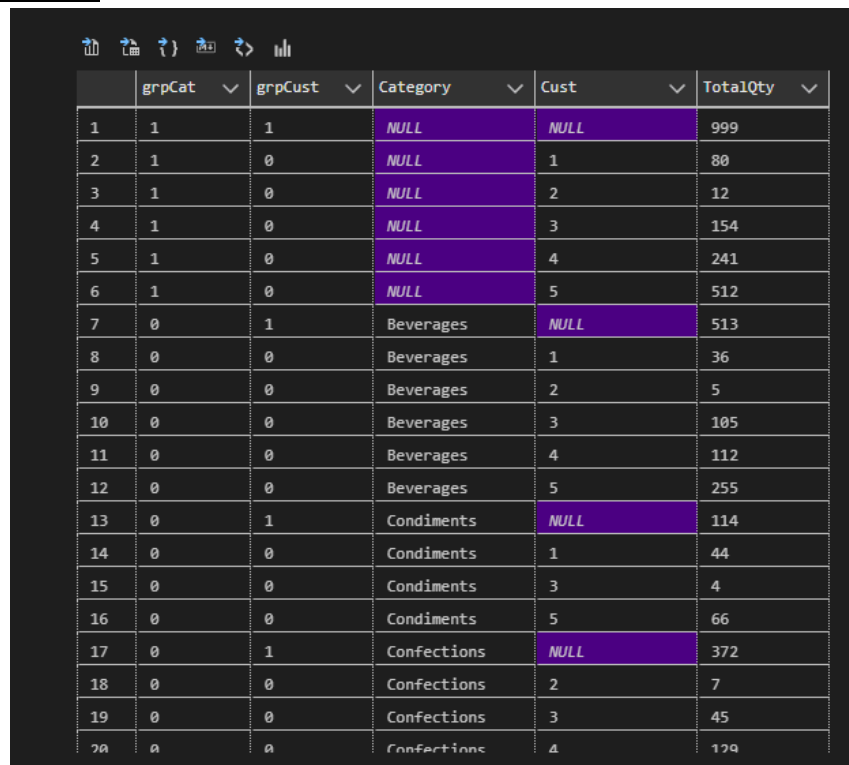
En este paso vamos a utilizar la función `GROUPING_ID`, que nos permite identificar si una fila del resultado corresponde a un grupo agregado (subtotal o total) dentro de una consulta que utiliza `CUBE`, `ROLLUP` o `GROUPING SETS`.

`GROUPING_ID` devuelve un número que indica cuáles columnas han sido agrupadas a nivel total (es decir, si contienen valores `NULL` generados por la agrupación). Cuanto mayor es el número, mayor es el nivel de agregación. Por ejemplo, si `GROUPING_ID(Category)` devuelve 1, significa que la fila representa un subtotal sin un valor específico de categoría.

### Código:

```
SELECT      GROUPING_ID(Category)AS grpCat, GROUPING_ID(Cust)
AS grpCust,
            Category, Cust, SUM(Qty) AS TotalQty
FROM Sales.CategorySales
GROUP BY CUBE(Category,Cust)
ORDER BY Category, Cust;
```

### Resultado:



	grpCat	grpCust	Category	Cust	TotalQty
1	1	1	NULL	NULL	999
2	1	0	NULL	1	80
3	1	0	NULL	2	12
4	1	0	NULL	3	154
5	1	0	NULL	4	241
6	1	0	NULL	5	512
7	0	1	Beverages	NULL	513
8	0	0	Beverages	1	36
9	0	0	Beverages	2	5
10	0	0	Beverages	3	105
11	0	0	Beverages	4	112
12	0	0	Beverages	5	255
13	0	1	Condiments	NULL	114
14	0	0	Condiments	1	44
15	0	0	Condiments	3	4
16	0	0	Condiments	5	66
17	0	1	Confections	NULL	372
18	0	0	Confections	2	7
19	0	0	Confections	3	45
20	0	0	Confections	4	129



## 2.7. Finalmente limpiar los cambios:

### **Explicación:**

En este último paso vamos a eliminar la vista Sales.CategorySales que creamos para los ejercicios anteriores. Esta acción forma parte de las buenas prácticas al trabajar con bases de datos, especialmente cuando se crean objetos temporales para consultas o laboratorios.

Antes de eliminarla, usamos la función OBJECT\_ID para verificar si la vista existe. Si existe, la instrucción DROP VIEW se ejecuta para eliminarla

### **Código:**

```
IF OBJECT_ID('Sales.CategorySales','V') IS NOT NULL DROP VIEW  
Sales.CategorySales  
GO
```

### **Resultado:**

Los comandos se han completado correctamente.

Tiempo total de ejecución: 00:00:00.015

**Ejercicio 03: Escribir consultas que usan las cláusulas GROUPING SETS, CUBE y ROLLUP**

**3.1. Escribir una sentencia SELECT contra la tabla Sales.Customers y traer las columnas country, city y una columna calculada noofcustomers como una cantidad de clientes. Traer multiples grouping sets basados en las columnas country y city, la columna country, la columna city, y una columna con un grouping set vacio:**

**Explicación:**

En este paso vamos a realizar una consulta sobre la tabla Sales.Customers para obtener un resumen de clientes agrupados por distintas combinaciones de ubicación. Mostraremos las columnas country y city, y además una columna calculada llamada noofcustomers, que representa la cantidad de clientes mediante la función COUNT(\*).

Utilizaremos la cláusula GROUPING SETS para definir varios niveles de agrupación dentro de una misma consulta. En este caso, pedimos los siguientes resúmenes:

- Agrupación por país y ciudad (detallado)
- Agrupación solo por país (subtotales por país)
- Agrupación solo por ciudad (subtotales por ciudad)
- Y finalmente, un total general (sin agrupar por ninguna columna)

Esto nos permite visualizar de forma clara y eficiente los distintos niveles de información sin necesidad de combinar múltiples consultas con UNION ALL.

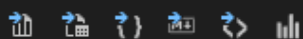
**Código:**

```
SELECT
    country,
    city,
    COUNT(*) AS noofcustomers
FROM Sales.Customers
GROUP BY GROUPING SETS (
    (country, city),
    (country),
    (city),
    ()
);
```

**Resultado:**

(160 filas afectadas)

Tiempo total de ejecución: 00:00:00.052



	country	city	noofcustomers
1	Germany	Aachen	1
2	NULL	Aachen	1
3	USA	Albuquerque	1
4	NULL	Albuquerque	1
5	USA	Anchorage	1
6	NULL	Anchorage	1
7	Denmark	Århus	1
8	NULL	Århus	1
9	Spain	Barcelona	1
10	NULL	Barcelona	1
11	Venezuela	Barquisimeto	1
12	NULL	Barquisimeto	1
13	Italy	Bergamo	1
14	NULL	Bergamo	1
15	Germany	Berlin	1
16	NULL	Berlin	1
17	Switzerland	Bern	1
18	NULL	Bern	1
19	USA	Boise	1
20	NULL	Boise	1

### 3.2. Escribir una sentencia SELECT contra la vista Sales.OrderValues y traer las columnas:

- Año de la columna orderdate como orderyear.
- Mes de la columna orderdate como ordermonth.
- Día de la columna orderdate como orderday.
- Valor de Total de Ventas utilizando la columna val como salesvalue.
- Devolver todas los posibles grouping sets basados en las columnas orderyear, ordermonth, y orderday.

#### Explicación:

En este paso vamos a consultar la vista Sales.OrderValues para obtener un resumen de los valores de ventas agrupados por fecha. Específicamente, vamos a extraer tres componentes de la columna orderdate: el año (orderyear), el mes (ordermonth) y el día (orderday). Además, calcularemos el total de ventas usando la columna val, a la que llamaremos salesvalue.

Para generar todos los posibles niveles de agregación entre año, mes y día, usaremos la cláusula GROUP BY CUBE. Esto nos permite obtener:

- Valores detallados por año, mes y día,
- Totales por año y mes (agrupando todos los días),
- Totales por año (agrupando todos los meses y días),
- Totales por mes y día, por mes, por día,
- Y también el total general de todas las ventas.

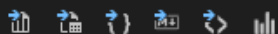
#### Código:

```
SELECT
    YEAR(orderdate) AS orderyear,
    MONTH(orderdate) AS ordermonth,
    DAY(orderdate) AS orderday,
    SUM(val) AS salesvalue
FROM Sales.OrderValues
GROUP BY CUBE (
    YEAR(orderdate),
    MONTH(orderdate),
    DAY(orderdate)
);
```

**Resultado:**

(948 filas afectadas)

Tiempo total de ejecución: 00:00:00.084



	orderyear ▾	ordermonth ▾	orderday ▾	salesvalue ▾
1	2007	1	1	6931.60
2	2008	1	1	1738.00
3	NULL	1	1	8669.60
4	2007	4	1	851.20
5	2008	4	1	11549.89
6	NULL	4	1	12401.09
7	2007	5	1	5636.96
8	2008	5	1	5448.57
9	NULL	5	1	11085.53
10	2007	7	1	142.50
11	NULL	7	1	142.50
12	2006	8	1	1424.00
13	2007	8	1	2697.50
14	NULL	8	1	4121.50
15	2007	9	1	716.64
16	NULL	9	1	716.64
17	2006	10	1	240.40
18	2007	10	1	3633.10
19	NULL	10	1	3873.50
20	2006	11	1	2296.00

### 3.3. Copiar la consulta previa y modificarla para utilizar la cláusula ROLLUP en vez de CUBE.:

#### Explicación:

En este paso vamos a tomar la consulta anterior y modificarla para que, en lugar de usar CUBE, utilice la cláusula ROLLUP. Seguimos trabajando con la vista Sales.OrderValues, extrayendo el año, mes y día de la columna orderdate, y sumando el valor total de ventas a través de la columna val, renombrada como salesvalue.

La diferencia clave es que ROLLUP genera los totales en orden jerárquico, es decir, en este caso primero agrupa por día, luego por mes y finalmente por año. Esto nos permite ver:

- Los valores detallados por año, mes y día,
- Luego subtotales mensuales dentro de cada año (sin el día),
- Luego subtotales anuales (sin mes ni día),
- Y al final, el total general para todos los datos.

A diferencia de CUBE, ROLLUP no combina todas las posibles agrupaciones, sino que sigue una secuencia lógica útil para reportes

#### Código:

```
SELECT
    YEAR(orderdate) AS orderyear,
    MONTH(orderdate) AS ordermonth,
    DAY(orderdate) AS orderday,
    SUM(val) AS salesvalue
FROM Sales.OrderValues
GROUP BY ROLLUP (
    YEAR(orderdate),
    MONTH(orderdate),
    DAY(orderdate)
);
```

**Resultado:**

(948 filas afectadas)

Tiempo total de ejecución: 00:00:00.084



	orderyear ▾	ordermonth ▾	orderday ▾	salesvalue ▾
1	2007	1	1	6931.60
2	2008	1	1	1738.00
3	NULL	1	1	8669.60
4	2007	4	1	851.20
5	2008	4	1	11549.89
6	NULL	4	1	12401.09
7	2007	5	1	5636.96
8	2008	5	1	5448.57
9	NULL	5	1	11085.53
10	2007	7	1	142.50
11	NULL	7	1	142.50
12	2006	8	1	1424.00
13	2007	8	1	2697.50
14	NULL	8	1	4121.50
15	2007	9	1	716.64
16	NULL	9	1	716.64
17	2006	10	1	240.40
18	2007	10	1	3633.10
19	NULL	10	1	3873.50
20	2006	11	1	2296.00

### 3.4. Escribir una sentencia SELECT contra la vista Sales.OrderValues y traer las columnas:

- **Columna Calculada con el alias groupid (usar la función GROUPING\_ID con los campos order year y order month como parametros).**
- **Año de la columna orderdate como orderyear.**
- **Mes de la columna orderdate como ordermonth.**
- **Valor Total de Ventas utilizando la columna val como salesvalue.**
- **Desde Año y mes month forman una jerarquía, retornar todos los grouping sets interesantes basados en las columnas orderyear y ordermonth y ordenar el resultado por groupid, orderyear, and ordermonth**

#### **Explicación:**

En este paso vamos a consultar nuevamente la vista Sales.OrderValues, pero esta vez enfocándonos en una jerarquía temporal entre el año y el mes del campo orderdate. Queremos resumir el total de ventas (salesvalue) y además identificar claramente el nivel de agregación de cada fila usando la función GROUPING\_ID.

Creamos una columna calculada llamada groupid, que usa GROUPING\_ID(YEAR(orderdate), MONTH(orderdate)) para asignar un número que representa el nivel de agrupación. Esta función nos permite saber si la fila representa datos detallados (año y mes), un subtotal por año, o un total general.

Luego, usamos GROUPING SETS para definir tres niveles de agrupación:

- Por año y mes (datos detallados),
- Solo por año (subtotales anuales),
- Y un grupo vacío para el total general.

Finalmente, ordenamos los resultados por groupid, orderyear y ordermonth para que los datos se muestren de forma clara y jerárquica



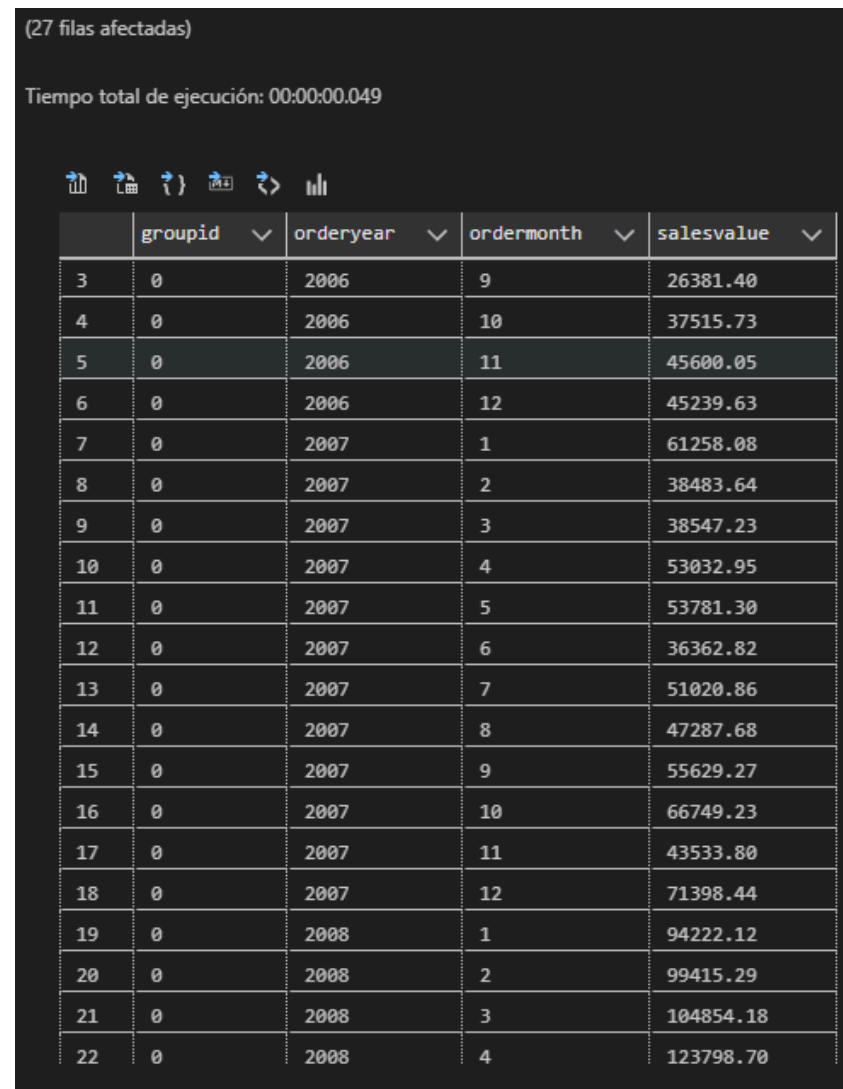
**Código:**

```
SELECT
  GROUPING_ID(YEAR(orderdate), MONTH(orderdate)) AS groupid,
  YEAR(orderdate) AS orderyear,
  MONTH(orderdate) AS ordermonth,
  SUM(val) AS salesvalue
FROM Sales.OrderValues
GROUP BY GROUPING SETS (
  (YEAR(orderdate), MONTH(orderdate)),
  (YEAR(orderdate)),
  ()
)
ORDER BY groupid, orderyear, ordermonth;
```

**Resultado:**

(27 filas afectadas)

Tiempo total de ejecución: 00:00:00.049



	groupid	orderyear	ordermonth	salesvalue
3	0	2006	9	26381.40
4	0	2006	10	37515.73
5	0	2006	11	45600.05
6	0	2006	12	45239.63
7	0	2007	1	61258.08
8	0	2007	2	38483.64
9	0	2007	3	38547.23
10	0	2007	4	53032.95
11	0	2007	5	53781.30
12	0	2007	6	36362.82
13	0	2007	7	51020.86
14	0	2007	8	47287.68
15	0	2007	9	55629.27
16	0	2007	10	66749.23
17	0	2007	11	43533.80
18	0	2007	12	71398.44
19	0	2008	1	94222.12
20	0	2008	2	99415.29
21	0	2008	3	104854.18
22	0	2008	4	123798.70

### 3. Conclusiones

---

- Se comprendió la estructura básica de un proyecto Web API en .NET, incluyendo la creación de modelos, servicios y controladores para organizar el código de manera clara y reutilizable.
  - Swagger resultó ser una herramienta muy útil para probar y documentar las rutas de la API, permitiendo validar fácilmente cada operación sin necesidad de herramientas externas.
  - El uso del modificador required en .NET 7+ ayuda a prevenir errores en tiempo de compilación al asegurar que las propiedades esenciales no queden sin asignar.
- 

### 4. Referencias Bibliográficas

---

- Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387.
- Elmasri, R., & Navathe, S. (2015). Fundamentals of Database Systems (7th ed.). Pearson.
- Coronel, C., & Morris, S. (2017). Database Systems: Design, Implementation, and Management. Cengage Learning.