

Universidad Nacional del Nordeste



Facultad de Ciencias Exactas y Naturales y Agrimensura
Licenciatura en Sistemas de Información
Base de datos I
Año: 2023

Anexo: Integración de proyectos

Comisión 1 grupo 5

Alumno: Bournisent, Matias

L.U. N°: 50.434

D.N.I. N°: 39.126.075

Alumno: Comba, Carlos Alfredo

L.U. N°: 44.055

D.N.I. N°: 32.335.478

Alumno: Ramirez Gonzalo Daniel

L.U. N°: 56.838

D.N.I. N°: 44.543.439

Alumno: Zini, Franco Joaquin

L.U. N°: 50.717

D.N.I. N°: 40.049.028

Anexo

Grupo 1 Procedimientos y funciones almacenadas

El tema investigado por el grupo 1 el cual integramos a nuestro trabajo fue sobre dos herramientas esenciales para la gestión de bases de datos: los procedimientos almacenados y las funciones definidas por el usuario en SQL Server. Estas herramientas permiten optimizar y simplificar las operaciones de manipulación de datos, así como reutilizar código y mejorar la seguridad. El objetivo de su trabajo fue proporcionar conocimiento básico sobre cómo crear y utilizar estos elementos en su proyectos, con el fin de aumentar la eficiencia y la productividad.

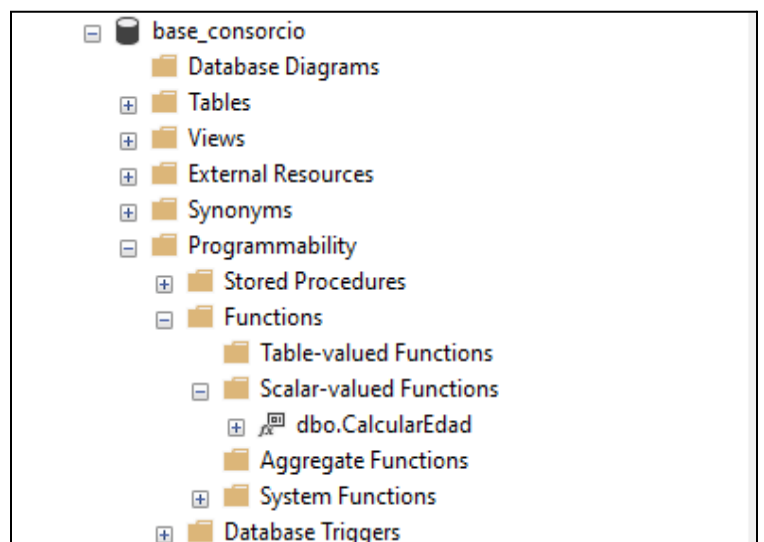
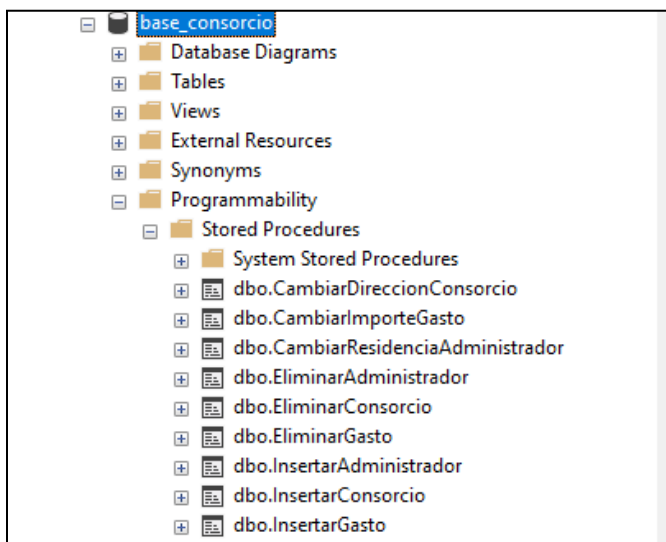
un integrante del grupo tuvo un error ocasionado posiblemente por la version del sql server ([sql server 12.0.2269](#)), DROP PROCEDURE IF EXISTS [nombre_procedimiento] Esta versión de código en la versión de motor no funciona, se agregó al script un código que funciona de la misma manera pero no tira error

```
IF OBJECT_ID(' nombre_procedimiento ', 'P') IS NOT NULL
```

```
    DROP PROCEDURE nombre_procedimiento;
```

se hizo la modificación para poder seguir.

Se pudo implementar todos los script y probar el funcionamientos de los mismos, en la siguiente imagen se pueden observar los procedimientos almacenados y las funciones conviviendo en la base de datos, base consorcio



Conclusión:

Por la integración que hicimos y lo pudimos entender es que un procedimiento almacenado es un conjunto de instrucciones que se ejecutan en el servidor de bases de datos y pueden aceptar y devolver parámetros, realizar operaciones, devolver un valor de estado y llamar a otros procedimientos. Una función definida por el usuario es una rutina que acepta parámetros, realiza una acción y devuelve un valor escalar o una tabla.

Ambas herramientas tienen ventajas como la simplificación del código, la reutilización, la optimización del rendimiento, la modularidad y la seguridad.

Grupo 2 Triggers

El tema investigado por el grupo 2 el cual integramos a nuestro trabajo fue sobre el tema de los triggers en SQL Server, unos objetos que se ejecutan automáticamente cuando se producen ciertos eventos en las tablas de una base de datos. Detallan qué son los triggers, cómo se crean y se utilizan, y qué beneficios aportan para automatizar acciones específicas en las tablas, como verificar restricciones, actualizar información o sincronizar datos.

G2-BDI-Triggers.sql, modificaciones al script del grupo 2 para evitar errores

```
IF OBJECT_ID('auditoriaAdministrador', 'U') IS NOT NULL
```

```
DROP TABLE auditoriaAdministrador;
```

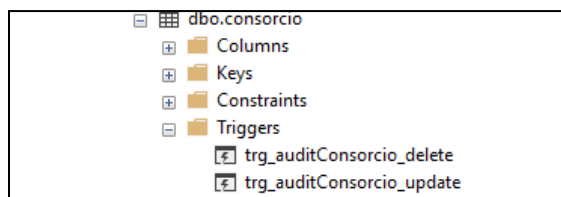
```
GO
```

```
IF OBJECT_ID('trg_auditGasto_delete', 'TR') IS NOT NULL
```

```
DROP TRIGGER trg_auditGasto_delete;
```

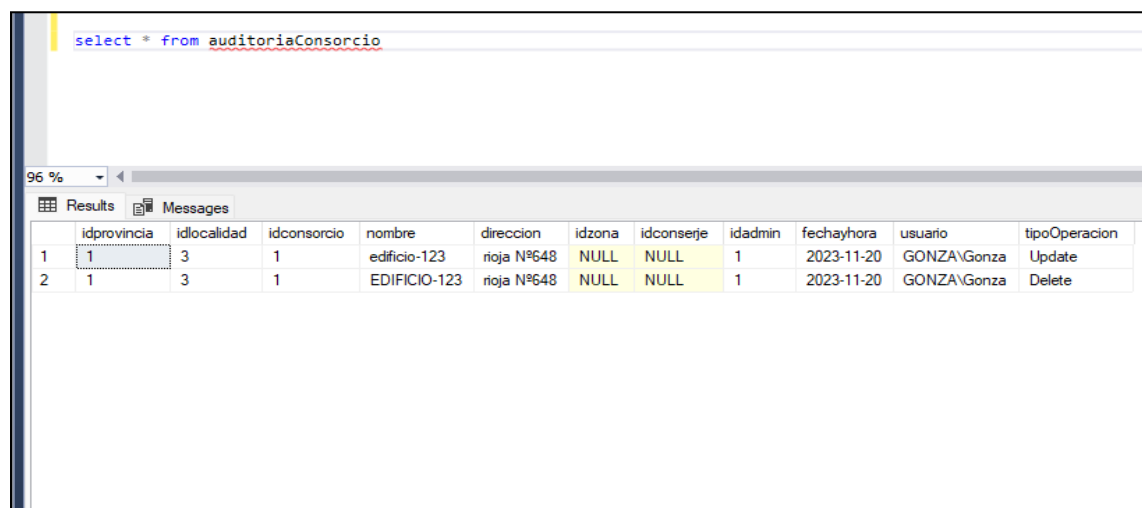
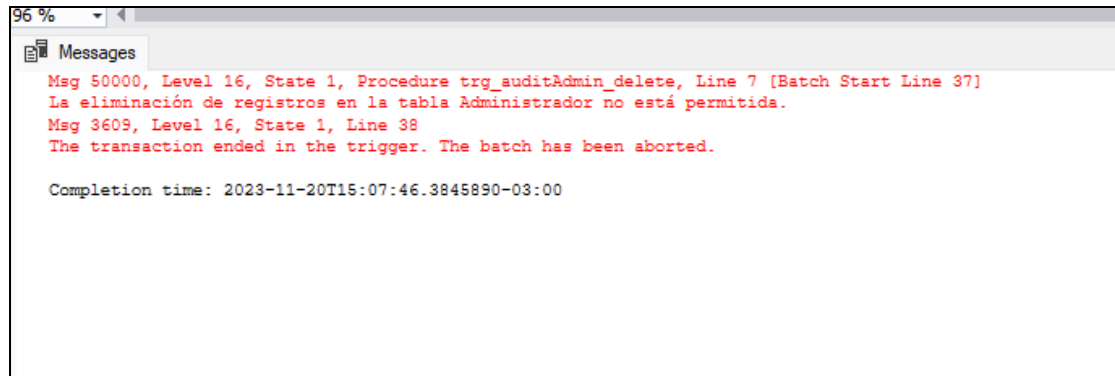
```
GO
```

Se insertaron los triggers correspondientes a las tablas consorcio, administrador y gasto. Podemos visualizar en el Management Studio los triggers asociados a estas tablas, como por ejemplo:



Se realizó la prueba de eliminar un administrador y comprobar el correcto funcionamiento del disparador:

Uno de los triggers implementados se ejecutará después de que se haya realizado la actualización en la tabla realizando una inserción en la tabla auditoriaConsorcio de los datos antiguos, la fecha de la actualización y el nombre de usuario que realizó la operación. ejecutamos un update para comprobar el funcionamiento del TRIGGER y se puede comprobar los registros modificados.



Conclusión:

Por la integración que hicimos y lo pudimos entender es que los triggers son muy útiles para optimizar la gestión de bases de datos, mejorar la seguridad e integridad de los datos y ahorrar tiempo y recursos.

Grupo 4 : Manejo de transacciones y transacciones anidadas

El tema investigado por el grupo 4 el cual integramos a nuestro trabajo fue sobre el tema de transacciones en SQL Server, que son unidades de trabajo que se ejecutan cuando se modifican los datos de una base de datos. Explican cómo funcionan las transacciones, qué propiedades deben cumplir, qué estados pueden tener y cómo se pueden iniciar, confirmar o revertir. También investigaron sobre los diferentes tipos de opciones que existen para aplicar las transacciones, como las transacciones anidadas, que son transacciones dentro de otras transacciones.

Hicimos las pruebas en base al script proporcionado por el grupo 4 para poder verificar el funcionamiento del mismo. Utilizamos transacción con error en la ejecución y sin error:

```
SET LANGUAGE Spanish;

BEGIN TRY
    BEGIN TRAN

        --Inserción registro administrador
        INSERT INTO administrador(apeynom,viveahi,tel,sexo,fechnac)
        values ('Juan Pablo Duete', 'S', '3794000102', 'M', '19890625')

        --Inserción registro consorcio
        INSERT INTO consorcio(idprovincia,idlocalidad,idconsorcio,
        Nombre,direccion,idzona,idconserje,idadmin)
        VALUES (7, 7, 1, 'EDIFICIO-24500', '9 de julio N° 1650', 2,
        Null, (select top 1 idadmin from administrador order by idadmin desc))
        --id del último administrador

        --Inserción 3 registros gasto
        INSERT INTO gasto
        (idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
        VALUES (30, 7, 1,1,'20231005',2,5000.00)

        INSERT INTO gasto
        (idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
        VALUES (7, 7, 1,1,'20231015',2,20000.00)

        INSERT INTO gasto
        (idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
        VALUES (7, 7, 1,1,'20231028',2,10000.00)

        COMMIT TRAN
    END TRY

BEGIN CATCH
    SELECT ERROR_MESSAGE() As Error-- Se captura el error y lo muestra.
    ROLLBACK TRAN
END CATCH
```

```
SET LANGUAGE Spanish;

BEGIN TRY
    BEGIN TRAN

        --Inserción registro administrador
        INSERT INTO administrador(apeynom,viveahi,tel,sexo,fechnac)
        values ('Juan Pablo Duete', 'S', '3794000102', 'M', '19890625')

        --Inserción registro consorcio
        INSERT INTO consorcio(idprovincia,idlocalidad,idconsorcio,
        Nombre,direccion,idzona,idconserje,idadmin)
        VALUES (7, 7, 1, 'EDIFICIO-24500', '9 de julio N° 1650', 2,
        Null, (select top 1 idadmin from administrador order by idadmin desc))
        --id del último administrador

        --Inserción 3 registros gasto
        INSERT INTO gasto
        (idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
        VALUES (7, 7, 1,1,'20231005',2,5000.00)

        INSERT INTO gasto
        (idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
        VALUES (7, 7, 1,1,'20231015',2,20000.00)

        INSERT INTO gasto
        (idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
        VALUES (7, 7, 1,1,'20231028',2,10000.00)

        COMMIT TRAN
    END TRY

BEGIN CATCH
    SELECT ERROR_MESSAGE() As Error-- Se captura el error y lo muestra.
    ROLLBACK TRAN
END CATCH
```

Analizamos el funcionamiento de las transacciones anidadas t cómo estas transacciones permiten operaciones dentro de otras operaciones, lo que ofrece mayor control y precisión en la gestión de datos.

```

SET LANGUAGE Spanish;

BEGIN TRY
    BEGIN TRAN TS_Anidades
        SELECT CONCAT('El nivel de anidamiento es ', @@TRANCOUNT) As 'Numero de
anidamientos' --Esta sentencia cuenta el número de transacciones anidades
        BEGIN TRAN TS_InsertarAdmin
            --Inserción registro administrador
            SELECT CONCAT('El nivel de anidamiento es ', @@TRANCOUNT) As
'Numero de anidamientos'
            INSERT INTO administrador(apeynom,viveahi,tel,sexo,fechnac)
values ('Juan José Paso', 'S', '3794000102', 'M', '19890625')
            COMMIT TRAN TS_InsertarAdmin
            BEGIN TRAN TS_InsertarConsortio
                --Inserción registro consorcio
                SELECT CONCAT('El nivel de anidamiento es ', @@TRANCOUNT) As
'Numero de anidamientos'
                INSERT INTO consorcio(idprovincia,idlocalidad,idconsorcio,
Nombre,direccion,idzona,idconserje,idadmin)
VALUES (7, 7, 2, 'EDIFICIO-24500', '9 de julio N° 1650', 2,
Null, (select top 1 idadmin from administrador order by idadmin desc))
                --Para insertar el id del último administrador se hace un
select del ultimo id de administrador insertado.

                BEGIN TRAN TS_InsertarGastos
                    SELECT CONCAT('El nivel de anidamiento es ',
@@TRANCOUNT) As 'Numero de anidamientos'
                    --Inserción 3 registros de gastos
                    INSERT INTO gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
VALUES (7, 7, 2,1,'20231005',2,5000.00)

                    INSERT INTO gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
VALUES (7, 7, 2,1,'20231015',2,20000.00)

                    INSERT INTO gasto
(idprovincia,idlocalidad,idconsorcio,periodo,fechapago,idtipogasto,importe)
VALUES (7, 7, 2,1,'20231028',2,10000.00)
                    COMMIT TRAN TS_InsertarGastos
                COMMIT TRAN TS_InsertarConsortio

            COMMIT TRAN TS_Anidades
        END TRY

    BEGIN CATCH
        SELECT ERROR_MESSAGE() As Error -- Se captura el error y lo muestra.
        ROLLBACK TRAN TS_Anidades
    END CATCH

```

Conclusión:

El manejo de transacciones en SQL Server representa una poderosa herramienta para la gestión de bases de datos. La capacidad de agrupar sentencias y ejecutarlas como un conjunto coherente, asegurando la integridad de los datos incluso en operaciones complejas, es una ventaja sustancial al trabajar con volúmenes masivos de información.

Grupo 6 : Manejo de permisos a nivel de usuarios de base de datos

Para verificar el trabajo realizado por el grupo 6 creamos el siguiente lote de pruebas para verificar los permisos de los dos usuarios creados:

```
--Pruebas para usuario usuarioDBA con rol DBA_Role--
USE base_consortio;
EXECUTE AS LOGIN = 'logDBA' ;
PRINT '(Ejecutando como usuario '+ USER +')'

PRINT 'Prueba 1: Crear o modificar tablas, deberia ser exitosa si la tabla no existe'
CREATE TABLE Ejemplo (ID INT, cantidad INT);
GO

PRINT 'Prueba 2: Insertar 2 registros en la tabla Ejemplo, deberia ser exitosa'
INSERT INTO Ejemplo (ID, cantidad) VALUES (1, 10);
INSERT INTO Ejemplo (ID, cantidad) VALUES (2, 20);
GO

PRINT 'Prueba 3: Actualizar un registro en la tabla Ejemplo, deberia ser exitosa'
UPDATE Ejemplo SET cantidad = 30 WHERE ID = 2;
GO
REVERT; -- Salir del contexto de usuarioDBA

-----2da prueba-----

--Pruebas para usuario usuarioReadOnly con rol ReadOnly_Role--
EXECUTE AS LOGIN = 'logRO'; -- Iniciar sesión como usuarioReadOnly
PRINT '(Ejecutando como usuario '+ USER +')'

PRINT 'Prueba 1: SELECT sobre tabla, deberia ser exitosa '
SELECT * FROM Ejemplo;
GO

PRINT 'Prueba 2: Insertar 2 registros en la tabla Ejemplo, deberia fallar'
INSERT INTO Ejemplo (ID, cantidad) VALUES (4, 100);
INSERT INTO Ejemplo (ID, cantidad) VALUES (6, 200);
GO

PRINT 'Prueba 3: Actualizar un registro en la tabla Ejemplo, deberia fallar'
UPDATE Ejemplo SET cantidad = 30 WHERE ID = 2;
GO
REVERT; -- Salir del contexto de usuarioReadOnly
```

Podemos ver que la salida es la esperada, por lo que los usuarios poseen los permisos adecuados

```
---Pruebas para usuario usuarioDBA con rol DBA_Role---
(Ejecutando como usuario usuarioDBA)
Prueba 1: Crear o modificar tablas, debería ser exitosa si la tabla no existe
Msg 2714, Level 16, State 6, Line 7
There is already an object named 'Ejemplo' in the database.
Prueba 2: Insertar 2 registros en la tabla Ejemplo, debería ser exitosa

(1 row affected)

(1 row affected)
Prueba 3: Actualizar un registro en la tabla Ejemplo, debería ser exitosa

(5 rows affected)
---Pruebas para usuario usuarioReadOnly con rol ReadOnly_Role---
(Ejecutando como usuario usuarioReadOnly)
Prueba 1: SELECT sobre tabla, debería ser exitosa

(10 rows affected)
Prueba 2: Insertar 2 registros en la tabla Ejemplo, debería fallar
Msg 229, Level 14, State 5, Line 31
The INSERT permission was denied on the object 'Ejemplo', database 'base_consortorio', schema 'dbo'.
Msg 229, Level 14, State 5, Line 32
The INSERT permission was denied on the object 'Ejemplo', database 'base_consortorio', schema 'dbo'.
Prueba 3: Actualizar un registro en la tabla Ejemplo, debería fallar
Msg 229, Level 14, State 5, Line 36
The UPDATE permission was denied on the object 'Ejemplo', database 'base_consortorio', schema 'dbo'.

Completion time: 2023-11-20T21:40:04.7740370-03:00
```

Conclusión: La correcta configuración de usuarios y permisos de una base de datos es una tarea fundamental para preservar la seguridad, privacidad e integridad de los datos almacenados en ella. Cada usuario deberá tener asignados los mínimos permisos que permitan realizar su tarea adecuadamente, es decir, por ejemplo, no asignarle permisos de escritura a un usuario que se supone que sólo debe realizar lecturas de los datos ya que un manejo incorrecto de sus credenciales podría comprometer a la base de datos.

Grupo 7 : Backup y restore. Backup en linea

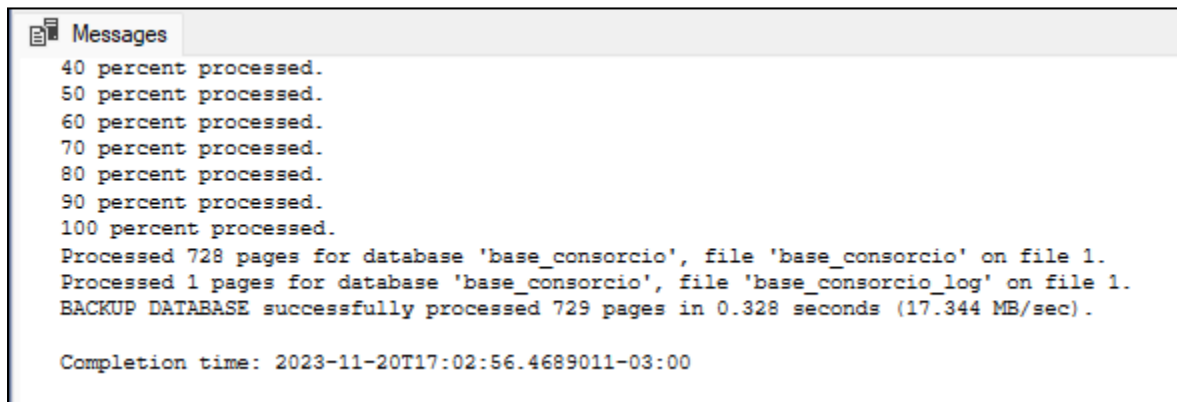
El tema investigado por el grupo 7 el cual integramos a nuestro trabajo fue sobre el tema de Backup y Restore de una base de datos, que son procesos que nos permiten crear y recuperar copias de seguridad de los datos almacenados en una base de datos. Han explicado la importancia de realizar copias de seguridad de forma regular y en distintos medios, para proteger los datos de posibles pérdidas, daños o eliminaciones. También investigaron cómo se pueden restaurar los datos de una copia de seguridad y volver a cargarlos en la base de datos, en caso de que se produzca algún problema o error. Detallaron los diferentes tipos de copias de seguridad, como las completas, las diferenciales, las incrementales o las continuas, y las opciones que se tiene para aplicarlas.

Realizamos el backup en forma local siguiendo el script del grupo 7, con el siguiente script:

```
Use base_consortio
DECLARE @Fecha VARCHAR(200)
SET @Fecha = REPLACE(CONVERT(VARCHAR,GETDATE()),':','.')
Declare @DireccionCarpeta Varchar(400)
Set @DireccionCarpeta = 'C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\Consortio\Consortio ' + @Fecha + '.bak'
BACKUP DATABASE base_consortio
TO DISK = @DireccionCarpeta
WITH INIT, NAME = 'base_consortio', STATS = 10
```

Nota: Cambiamos la ubicación del archivo al guardar.

Backup realizado correctamente:



The screenshot shows the 'Messages' window in SQL Server Enterprise Manager. It displays the progress of a backup operation for the 'base_consortio' database. The progress bar shows 100 percent processed. Below the progress bar, the following messages are displayed:

```
40 percent processed.
50 percent processed.
60 percent processed.
70 percent processed.
80 percent processed.
90 percent processed.
100 percent processed.
Processed 728 pages for database 'base_consortio', file 'base_consortio' on file 1.
Processed 1 pages for database 'base_consortio', file 'base_consortio_log' on file 1.
BACKUP DATABASE successfully processed 729 pages in 0.328 seconds (17.344 MB/sec).

Completion time: 2023-11-20T17:02:56.4689011-03:00
```

Nos da de resultado un archivo con la fecha actual y la hora en que se realizó el backup y su extensión .bak

Luego hicimos un DROP DATABASE base_consortio para verificar el funcionamiento de la restauración de la base de datos mediante el script para el restore:

```
DECLARE @NombreDataBase VARCHAR(200) = 'base_consortorio';
DECLARE @ubicacion NVARCHAR(128);

SELECT TOP 1 @ubicacion = m.physical_device_name
FROM msdb.dbo.backupset AS b
JOIN msdb.dbo.backupmediafamily AS m ON b.media_set_id = m.media_set_id
WHERE b.database_name = @NombreDataBase
AND RIGHT (m.physical_device_name, 4) = '.bak'
ORDER BY b.backup_start_date ASC;

RESTORE DATABASE base_consortorio
FROM DISK = @ubicacion
WITH REPLACE, RECOVERY;
```

96 %

Messages

Processed 728 pages for database 'base_consortorio', file 'base_consortorio' on file 1.
Processed 1 pages for database 'base_consortorio', file 'base_consortorio_log' on file 1.
RESTORE DATABASE successfully processed 729 pages in 0.089 seconds (63.920 MB/sec).
Completion time: 2023-11-20T17:17:38.6874415-03:00

Conclusión:

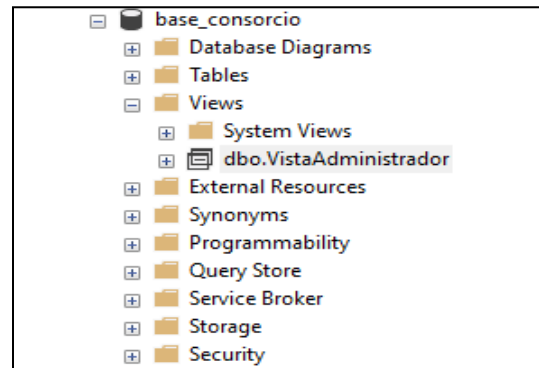
Hemos entendido el concepto y la importancia de Backup y Restore de una base de datos, que son procesos que nos permiten crear y recuperar copias de seguridad de los datos almacenados en una base de datos. Hemos visto los diferentes tipos de copias de seguridad, las opciones que tenemos para realizarlas y las situaciones en las que se deben utilizar. También hemos aprendido cómo se pueden restaurar los datos de una copia de seguridad y volver a cargarlos en la base de datos. Hemos comprendido que Backup y Restore de una base de datos son herramientas esenciales para garantizar la integridad y la disponibilidad de los datos en una base de datos.

Grupo 8 : Vistas y Vistas Indexadas

El tema investigado por el grupo 8 el cual integramos a nuestro trabajo fue sobre el tema de las vistas y las vistas indexadas en SQL, que son formas de mostrar una selección de datos de una o más tablas de una base de datos. Explican que las vistas nos permiten simplificar, personalizar y mejorar la seguridad de las consultas, limitando el acceso a los datos sensibles. Las vistas indexadas son vistas que tienen un índice asociado para mejorar el rendimiento de las consultas, ayudando al optimizador a encontrar los datos más rápido. Se analiza el concepto y la aplicación de las vistas y las vistas indexadas en SQL, así como los requisitos y las ventajas de usarlas.

Se realizó la vista vista sobre la tabla administrador que solo muestre los campos apeynom, sexo y fecha de nacimiento. Podemos visualizar las vistas creadas en nuestro explorador de archivos.

```
CREATE VIEW VistaAdministrador
AS SELECT apeynom, sexo, fechnac
FROM administrador;
go
```



Realizamos una inserción de datos sobre la vista recién creada para verificar como se comporta la misma.

Siguiente los pasos del script del grupo, se creó una vista que muestra los datos de las columnas de las siguientes tablas: Administrador->Apeynom, consorcio->Nombre, gasto->periodo, gasto->fechaPago, tipoGasto->descripcion

```
CREATE VIEW vista_adm_gasto_consortio WITH SCHEMABINDING AS
SELECT
    a.apeynom as nombre_admin,
    c.nombre as nombre_consortio,
    g.idgasto as idGasto,
    g.periodo as periodo,
    g.fechapago as fecha_pago,
    tp.descripcion as tipo_gasto
FROM
    dbo.administrador a
    JOIN dbo.consortio c ON a.idadmin = c.idadmin
    JOIN dbo.gasto g ON c.idadmin = g.idconsorcio
    JOIN dbo.tipogasto tp ON g.idtipogasto = tp.idtipogasto
GO
```

Luego verificamos su funcionamiento mediante la consulta:

```
select * from vista_adm_gasto_consortio;
```

Se creo un índice sobre la columna fechaPago sobre la vista recién creada

```
--index no agrupado  
CREATE NONCLUSTERED INDEX IDX_fechapago ON vista_adm_gasto_consortio(fecha_pago);
```

Conclusión:

Las vistas son tablas virtuales que representan una porción de los datos almacenados en una base de datos. Decimos que son virtuales debido a que no contienen datos reales, su contenido está definido por consultas que muestran una selección de columnas y registros de una o varias tablas, como también de otras vistas. Las vistas se producen de forma dinámica cuando se hace referencia a una de ellas.

Una vista indexada es una vista que tiene un índice asociado para acelerar el procesamiento de las consultas. Este índice se crea en la vista y no en las tablas base, ayudando al optimizador de consultas a encontrar los datos que necesita de forma eficiente cuando se consultan a través de la vista.