
Universidad Nacional Del Nordeste


Facultad de Ciencias Exactas y Naturales y Agrimensura

Licenciatura en Sistemas de Información

Catedra: Base de Datos 1

Año: 2022

# Proyecto Entrega Final: Proyecto Escuela Secundaria

Integrantes:

- Haberles, Lucas Francisco
- Luque Soler, Ignacio Gabriel
- Mulas, Vittorio Augusto
- Pasian Cominotti, Maurizio Giovanni
- Peloso, Nicolás Aníbal

# Índice

# Capítulo 1

## Introducción

### Tema

Dado que tenemos conocimiento del caso de nuestras antiguas escuelas secundarias de cómo se manejaban respecto a la administración de las mismas, nos proponemos como grupo de trabajo, unidos, trabajar sobre el caso práctico de la administración de una escuela del nivel secundario

### Definición o planteamiento del problema

En este proyecto tratamos el caso práctico de una escuela secundaria, en el cual incorporamos todos los conocimientos teóricos-prácticos obtenidos durante la cursada y también investigaciones en internet.

El proyecto tiene como finalidad la gestión de la administración de una escuela del nivel secundario, el sistema presentado en la continuidad de este informe debe permitir la gestión de alumnos, tutores, profesores, cursos y materias.

### Objetivos del Trabajo Práctico

El desarrollo del modelado de datos de nuestro proyecto <<*Escuela Secundaria*>> será capaz de; administrar alumnos que puedan tener varios tutores, tutores que puedan tener varios alumnos a cago, alumnos que pertenezcan a un solo curso, este último tiene asociadas varias materias y el alumno tiene tres notas asociadas a cada una de esas materias, éstas son dictadas por uno o varios profesores, los cuales se pueden encargar de varias materias de diferentes cursos.

# Capítulo 2
## Temas estudiados

### Funciones y procedimientos almacenados
#### Funciones
Las funciones en SQL pueden devolver siempre el mismo resultado (deterministas) o pueden retornar resultados distintos (no deterministas) cuando son llamadas con un mismo conjunto de valores de entrada.

A la hora de escribir un script complejo para SQL que realiza una determinada tarea, se puede crear una función para que se encargue de ello. Así, en caso de volver a necesitar nuevamente esa tarea, no habrá que volver a crear el script, siendo suficiente volver a llamar a esa función proporcionándole los parámetros necesarios.

#### Script de las funciones

```
USE proyectoEscuelaSecundaria
GO
IF OBJECT_ID (N'Edad', N'FN') IS NOT NULL
    DROP FUNCTION Edad;
GO
/*Devuelve la edad del alumno*/
CREATE FUNCTION Edad(@FechaNac date)
RETURNS int
AS
-- Returns the stock level for the product.
BEGIN
       RETURN DATEDIFF(YEAR, @FechaNac, GETDATE());
END;

IF OBJECT_ID (N'Aprueba', N'FN') IS NOT NULL
    DROP FUNCTION Aprueba;
GO

/*Si la nota es mayor o igual a 6 el alumno aprueba*/
CREATE FUNCTION Aprueba(@Nota decimal)
RETURNS BIT
AS
-- Returns the stock level for the product.
BEGIN
       IF @Nota >= 6
       BEGIN
              RETURN 1
       END

       RETURN 0
END;
```

#### Procedimientos almacenados
Un procedimiento almacenado (STORE PROCEDURE) está formado por un conjunto de instrucciones Transact-SQL que definen un determinado proceso a ejecutar, puede aceptar parámetros de entrada y devolver un valor o conjunto de resultados. Este procedimiento se guarda en el servidor y puede ser ejecutado en cualquier momento

Los procedimientos almacenados se crean mediante la sentencia CREATE PROCEDURE y se ejecutan con EXEC (o EXECUTE).

Para ejecutarlo también se puede utilizar el nombre del procedimiento almacenado sólo, siempre que sea la primera palabra del lote. Para eliminar un procedimiento almacenado utilizamos la sentencia DROP PROCEDURE.

### Diferencias

Por definición, las funciones son ideales para retornar valores con los que deseo operar, mientras que los procedimientos almacenados suelen ser la parte de salida final del código. Es decir que el procedimiento almacenado es el que se comunica directamente con el usuario y le presenta el resultado final.

Por eso, una función no puede invocar un procedimiento almacenado, solamente puede invocar otra función. En cambio, el procedimiento almacenado, como es código final, puede invocar funciones y procedimientos almacenados.

Así que a correcta integración de un sistema que use ambos, sería utilizar procedimientos almacenados para comunicarse y enviar datos a la aplicación; mientras que las funciones se usan para procesar datos internamente dentro del stored procedure.

### Script de los procedimientos almacenados

```
-- ============================================
-- Description:     <Agrega un alumno en la BD y en caso de tener un curso
asignado, se crean las tablas de las notas relacionadas a ese curso>
-- ============================================


USE proyectoEscuelaSecundaria
GO

CREATE PROCEDURE AgregarAlumno
@Nombre varchar(50),
@Apellido varchar(50),
@DNI int,
@Provincia int,
@Localidad int,
@Anio int,
@Division char,
@Sexo char,
@FechaNac date,
@Direccion varchar(255)
      AS
BEGIN

       IF @Anio IS NULL OR @Division IS NULL
              INSERT INTO Alumno
(Nombre,Apellido,Dni,Cod_provincia,cod_localidad,Anio,Division,Sexo,FechaNacimien
to,Direccion)
              VALUES
(@Nombre,@Apellido,@DNI,@Provincia,@Localidad,NULL,NULL,@Sexo,@FechaNac,@Direccio
n)
       ELSE
       BEGIN TRAN
              INSERT INTO Alumno
(Nombre,Apellido,Dni,Cod_provincia,cod_localidad,Anio,Division,Sexo,FechaNacimien
to,Direccion)
              VALUES
(@Nombre,@Apellido,@DNI,@Provincia,@Localidad,@Anio,@Division,@Sexo,@FechaNac,@Di
reccion)
              IF @@ERROR <> 0
              ROLLBACK TRAN
```

```
		DECLARE @codMateria int
		DECLARE materias CURSOR FOR
			SELECT Materia.Cod_Materia FROM Curso_Materia
			JOIN Materia ON Materia.Cod_Materia =
Curso_Materia.Cod_Materia
			WHERE Curso_Materia.Anio = @Anio AND Curso_Materia.Division =
@Division

		OPEN materias
		FETCH NEXT FROM materias INTO @codMateria
		WHILE @@FETCH_STATUS = 0
		BEGIN
			INSERT INTO Alumno_Nota (Dni, Cod_materia, Nota1, Nota2,
Nota3) VALUES(@DNI, @codMateria, NULL,NULL,NULL)
			IF @@ERROR <> 0
			ROLLBACK TRAN
			FETCH NEXT FROM materias INTO @codMateria

		END
		CLOSE materias
		DEALLOCATE materias
	COMMIT TRAN
END
--DROP PROCEDURE AgregarAlumno
--Para probar
/*EXECUTE AgregarAlumno
@Nombre='HÉCTOR',@Sexo='M',@Apellido='GORDILLO',@DNI=111,@Provincia=11,@Localidad
=26,@Direccion='General Viamonte 1658',@FechaNac='2008-4-2',@Anio=1,@Division='1'
SELECT * FROM Alumno WHERE Dni=111
DELETE FROM Alumno WHERE Dni=111*/

GO
-- =============================================
-- Author:		<Ignacio Luque>
-- Create date: <21-10-2022>
-- Description:	<Relaciona una materia con un curso y crea las notas
correspondientes a todos los alumnos de ese curso>
-- =============================================
CREATE PROCEDURE AgregarMateriaCurso
@Anio int,
@Division char,
@DNI int,
@Materia int
AS
BEGIN
	BEGIN TRAN
	INSERT INTO Curso_Materia (Anio, Division, DniProfesor, Cod_materia)
VALUES(@Anio, @Division, @DNI, @Materia)
	IF @@ERROR <> 0
		ROLLBACK TRAN

	DECLARE Alumnos CURSOR FOR SELECT Dni FROM Alumno WHERE Anio=@Anio AND
Division=@Division
	DECLARE @DniAlumno int

	OPEN Alumnos
	FETCH NEXT FROM Alumnos INTO @DniAlumno
	WHILE @@FETCH_STATUS = 0
	BEGIN
```

```
		INSERT INTO Alumno_Nota (Dni, Cod_materia) VALUES(@DniAlumno,
@Materia)
		IF @@ERROR <> 0
		ROLLBACK TRAN
		FETCH NEXT FROM Alumnos INTO @DniAlumno
	END
	CLOSE Alumnos
	DEALLOCATE Alumnos
	COMMIT TRAN
END

--DROP PROCEDURE AgregarMateriaCurso
--Para probar
/*
EXECUTE AgregarMateriaCurso @Anio=1,@Division=1,@DNI=34576049,@Materia=401
SELECT * FROM Curso_Materia WHERE DniProfesor=34576049 AND Cod_Materia=401 AND
Anio=1 AND Division='1'
SELECT * FROM Alumno JOIN Alumno_nota ON Alumno_nota.Dni=Alumno.Dni WHERE Anio=1
AND Division='1' AND Cod_materia=401*/

-- ============================================
-- Description:	<Elimina una materia de un curso y elimina las notas
relacionadas a esa materia correspondientes a todos los alumnos de ese curso>
-- ============================================
GO
CREATE PROCEDURE EliminarMateriaCurso
@Anio int,
@Division char,
@DNI int,
@Materia int
AS
BEGIN

	DECLARE Alumnos CURSOR FOR SELECT Dni FROM Alumno WHERE Anio=@Anio AND
Division=@Division
	DECLARE @DniAlumno int

	OPEN Alumnos
	FETCH NEXT FROM Alumnos INTO @DniAlumno
	WHILE @@FETCH_STATUS = 0
	BEGIN
		/*Primero borramos los registros de la tabla Alumno_Nota para
evitar el error por restricción de FK*/
		DELETE FROM Alumno_Nota WHERE Dni=@DniAlumno AND
Cod_materia=@Materia
		FETCH NEXT FROM Alumnos INTO @DniAlumno
	END
	DELETE FROM Curso_Materia WHERE Anio=@Anio AND Division=@Division AND
DniProfesor=@DNI AND Cod_materia=@Materia
	CLOSE Alumnos
	DEALLOCATE Alumnos
END

--Para probar
/*EXECUTE EliminarMateriaCurso @Anio=1,@Division=1,@DNI=34576049,@Materia=401
SELECT * FROM Alumno JOIN Alumno_nota ON Alumno_nota.Dni=Alumno.Dni WHERE Anio=1
AND Division='1' AND Cod_materia=401*/
```

## Disparadores

Un "trigger" (disparador o desencadenador) es un tipo de procedimiento almacenado que se ejecuta cuando se intenta modificar los datos de una tabla (o vista).

- Se definen para una tabla (o vista) específica.
- Se crean para conservar la integridad referencial y la coherencia entre los datos entre distintas tablas.
- Si se intenta modificar (agregar, actualizar o eliminar) datos de una tabla en la que se definió un disparador para alguna de estas acciones (inserción, actualización y eliminación), el disparador se ejecuta (se dispara) en forma automática.
- Un trigger se asocia a un evento (inserción, actualización o borrado) sobre una tabla.

La diferencia con los procedimientos almacenados del sistema es que los triggers:

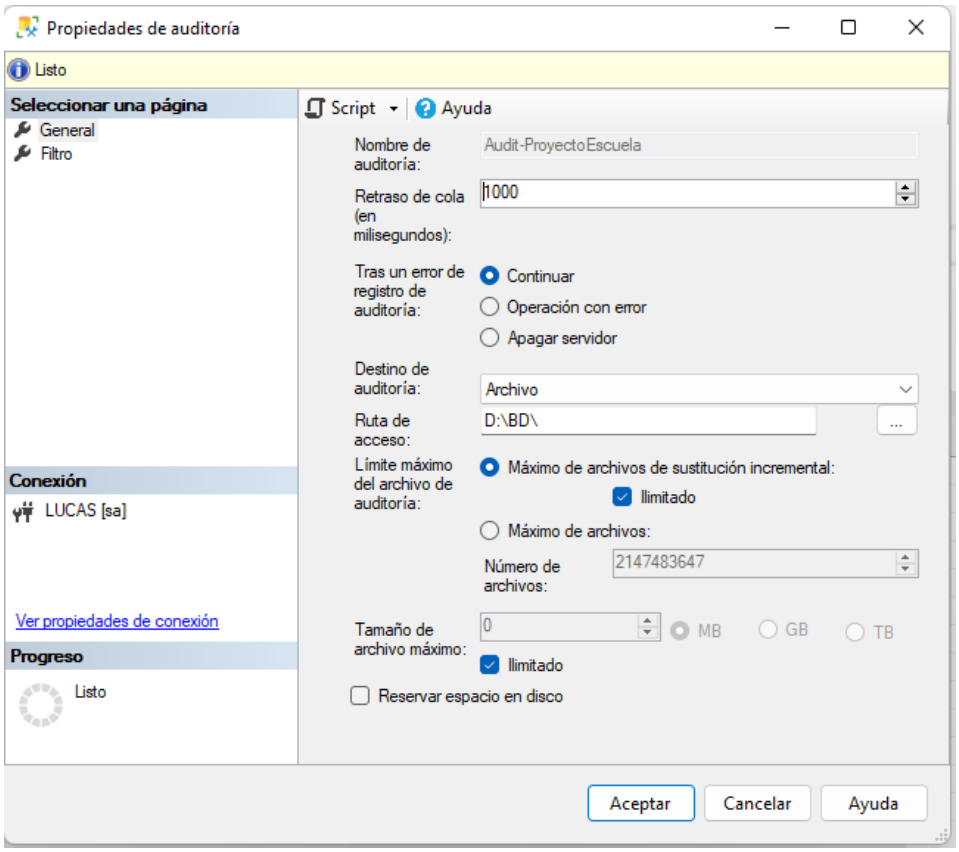
- no pueden ser invocados directamente; al intentar modificar los datos de una tabla para la que se ha definido un disparador, el disparador se ejecuta automáticamente.
- No reciben y retornan parámetros.
- son apropiados para mantener la integridad de los datos, no para obtener resultados de consultas.

Los disparadores, a diferencia de las restricciones "Check", pueden hacer referencia a campos de otras tablas.

### Disparadores que permiten auditar

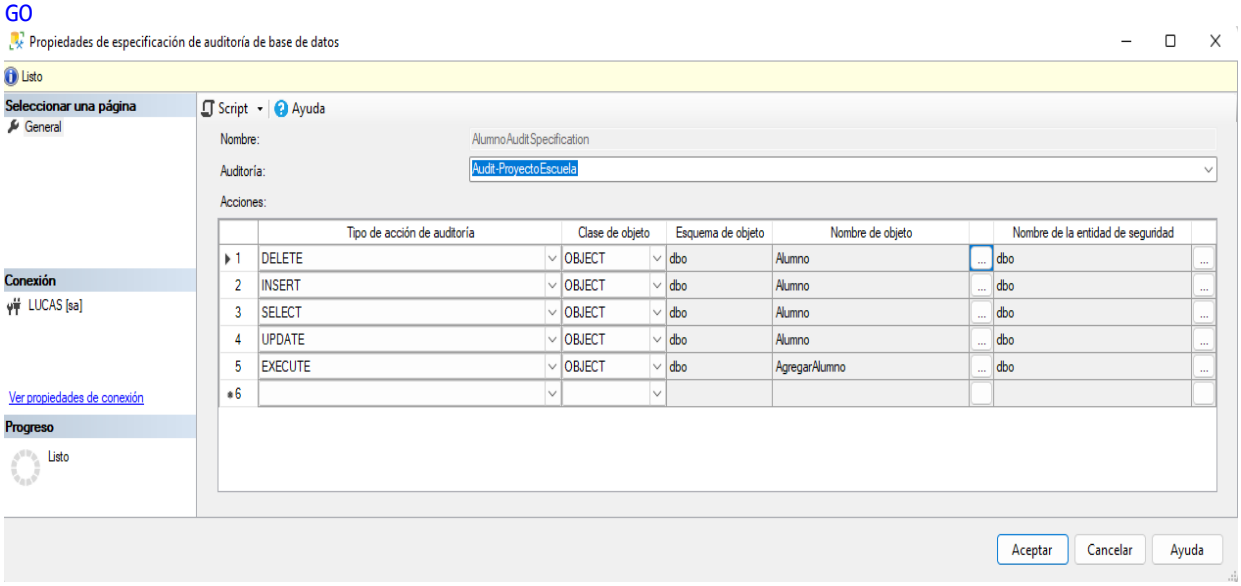
Para poder realizar auditorías en nuestra base de datos es necesario ejecutar estos script's

```
--1) Creamos la auditoría en el motor de bd, la cual almacenaremos en el disco D:
en la carpeta (anteriormente creada) /BD
USE [master]
GO
CREATE SERVER AUDIT [Audit-ProyectoEscuela]
TO FILE
(    FILEPATH = N'D:\BD'
    ,MAXSIZE = 0 MB
    ,MAX_ROLLOVER_FILES = 2147483647
    ,RESERVE_DISK_SPACE = OFF
) WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE)
GO
```




```
--2) indicamos que tipo de auditoria deberá realizar (SELECT, UPDATE, DELETE,
INSERT, EXECUTE), todos sobre la tabla principal [dbo].[Alumno]
USE [proyectoEscuelaSecundaria]
GO
CREATE DATABASE AUDIT SPECIFICATION [AlumnoAuditSpecification]
FOR SERVER AUDIT [Audit-ProyectoEscuela]
ADD (SELECT ON OBJECT::[dbo].[Alumno] BY [dbo]),
ADD (UPDATE ON OBJECT::[dbo].[Alumno] BY [dbo]),
ADD (DELETE ON OBJECT::[dbo].[Alumno] BY [dbo]),
ADD (INSERT ON OBJECT::[dbo].[Alumno] BY [dbo]),
ADD (EXECUTE ON OBJECT::[dbo].[AgregarAlumno] BY [dbo])

GO
```



## Triggers creados para nuestra BD

```
-- Description:      <Cambia el curso de un alumno, actualizando también sus
notas. Se dan tres situaciones:
--1-El nuevo curso sea válido y el alumno ya haya pertenecido a un curso, por lo
que se borran sus notas anteriores y se cargan las notas correspondientes al
nuevo curso
--2-El nuevo curso sea válido y el alumno no haya pertenecido a un curso, por lo
que se cargan las notas correspondientes al nuevo curso
--3-El curso sea NULL y el alumno ya haya pertenecido a un curso, por lo que se
borran las notas correspondientes al curso antiguo
-->
-- ============================================

CREATE TRIGGER UpdateAlumnoCurso ON Alumno
INSTEAD OF UPDATE
AS
BEGIN
	DECLARE @anioViejo int = (SELECT Anio FROM deleted)
	DECLARE @anioNuevo int = (SELECT Anio FROM inserted)
	DECLARE @divisionVieja char = (SELECT Division FROM deleted)
	DECLARE @divisionNueva char = (SELECT Division FROM inserted)
	DECLARE @dniAlumno int = (SELECT dni FROM inserted)

	BEGIN TRAN
	IF @anioNuevo IS NOT NULL AND @divisionNueva IS NOT NULL
		BEGIN
			UPDATE Alumno SET Anio=@anioNuevo, Division=@divisionNueva
WHERE Dni=@dniAlumno
			IF @@ERROR <> 0
			ROLLBACK TRAN

			DECLARE materias CURSOR FOR SELECT Cod_materia FROM
Curso_Materia WHERE Anio=@anioNuevo AND Division=@divisionNueva
			DECLARE @codMateria int
			OPEN materias
			FETCH NEXT FROM materias INTO @codMateria
			/*Si el alumno ya pertenecia a un curso debemos borrar sus
notas y asignarle nuevas*/
			IF @anioViejo IS NOT NULL AND @divisionVieja IS NOT NULL
			BEGIN
				DELETE FROM Alumno_Nota WHERE Dni=@dniAlumno
				IF @@ERROR <> 0
				ROLLBACK TRAN
			END
			/*Si el alumno no pertenecía a un curso, solo se deben
agregar las notas del curso al que se agregó*/
			WHILE @@FETCH_STATUS = 0
			BEGIN
				INSERT INTO Alumno_Nota (Dni,Cod_materia)
VALUES(@dniAlumno, @codMateria)
				IF @@ERROR <> 0
				ROLLBACK TRAN

				FETCH NEXT FROM materias INTO @codMateria
			END
			CLOSE materias
			DEALLOCATE materias
		END
	ELSE
	/*Si llega hasta aquí quiere decir que anio y/o division son NULL, por lo
que si el alumno ya pertenecía a un curso debemos borrar sus notas*/
	BEGIN
```

```
        UPDATE Alumno SET Anio=NULL, Division=NULL WHERE Dni=@dniAlumno
        IF @@ERROR <> 0
        ROLLBACK TRAN

                IF @anioViejo IS NOT NULL AND @divisionVieja IS NOT NULL
                        BEGIN
                                DELETE FROM Alumno_Nota WHERE Dni=@dniAlumno
                                IF @@ERROR <> 0
                                ROLLBACK TRAN
                        END
        END
        COMMIT TRAN
END

SELECT * FROM Alumno  JOIN Alumno_Nota ON Alumno_Nota.Dni=Alumno.Dni WHERE
Alumno.Dni=50350858
UPDATE Alumno SET Anio=1, Division='1' WHERE Dni=50350858
```

## Transacciones

Una transacción es un conjunto de operaciones que van a ser tratadas como una única unidad. Estas transacciones deben cumplir 4 propiedades fundamentales comúnmente conocidas como ACID (atomicidad, coherencia, asilamiento y durabilidad).

La transacción más simple en SQL Server es una única sentencia SQL.

Cuando enviamos este tipo de sentencia al SQL Server se escribe en el fichero de transacciones lo que va a ocurrir y a continuación realiza los cambios necesarios en la base de datos. Si hay algún tipo de problema al hacer esta operación el SQL Server puede leer en el fichero de transacciones lo que se estaba haciendo y si es necesario puede devolver la base de datos al estado en el que se encontraba antes de recibir la sentencia.

## Vistas

Una vista es una alternativa para mostrar datos de varias tablas. Una vista es como una tabla virtual que almacena una consulta. Los datos accesibles a través de la vista no están almacenados en la base de datos como un objeto.

Entonces, una vista almacena una consulta como un objeto para utilizarse posteriormente. Las tablas consultadas en una vista se llaman tablas base. En general, se puede dar un nombre a cualquier consulta y almacenarla como una vista.

Una vista suele llamarse también tabla virtual porque los resultados que retorna y la manera de referenciarlas es la misma que para una tabla.

Las vistas permiten:

- Ocultar información: permitiendo el acceso a algunos datos y manteniendo oculto el resto de la información que no se incluye en la vista. El usuario opera con los datos de una vista como si se tratara de una tabla, pudiendo modificar tales datos.
- Simplificar la administración de los permisos de usuario: se pueden dar al usuario permisos para que solamente pueda acceder a los datos a través de vistas, en lugar de concederle permisos para acceder a ciertos campos, así se protegen las tablas base de cambios en su estructura.
- Mejorar el rendimiento: se puede evitar “tipear” instrucciones repetidamente almacenando en una vista el resultado de una consulta compleja que incluya información de varias tablas.

Podemos crear vistas con: un subconjunto de registros y campos de una tabla; una unión de varias tablas; una combinación de varias tablas; un resumen estadístico de una tabla; un subconjunto de otra vista, combinación de vistas y tablas.

## Script de las vistas

```
-- Vistas creación alumnos por curso
CREATE VIEW Alumnos_curso with encryption
as
/*Utiliza la función edad para calcular la edad del alumno*/
SELECT  alumno.dni AS 'DNI', apellido AS 'Apellido', Alumno.nombre AS 'Nombre',
anio AS 'Año', division AS 'División', dbo.Edad(FechaNacimiento) AS 'Edad'
,provincia.Nombre AS 'Provincia', localidad.Nombre AS 'Localidad' from Alumno
       JOIN Localidad ON Localidad.Cod_provincia = Alumno.Cod_provincia AND
Localidad.Cod_localidad = Alumno.Cod_localidad
       JOIN Provincia ON Provincia.Cod_provincia = Alumno.Cod_provincia

-- Vistas creación promedios por materia
CREATE VIEW Promedios_alumnos with encryption
as
SELECT alumno.dni, Alumno.apellido, Alumno.nombre, anio, division, Materia.Nombre
AS 'Materia', Alumno_Nota.Nota1, Alumno_Nota.Nota2, Alumno_Nota.Nota3,
round((nota1+nota2+nota3)/3, 2) as "Promedio"
FROM alumno
 JOIN Alumno_Nota ON alumno.Dni=Alumno_Nota.Dni
 JOIN Materia ON Materia.Cod_materia = Alumno_Nota.Cod_materia


--Manipulación de la tabla vista
Declare @anio int,
            @division char;
--SE ASIGNA UN VALOR AL AÑO Y DIVISION PARA OBTENER LOS ALUMNOS DE UN DETERMINADO
CURSO.
set @anio = 1
set @division = 1
--se obtienen los alumnos del curso
select apellido, Nombre, dni
from Alumnos_curso
where anio= @anio and division = @division
order by Apellido;

/*Trae todos los alumnos cuyo promedio es mayor que 6 haciendo uso de la función
aprueba*/
SELECT * FROM Promedios_alumnos WHERE dbo.Aprueba(Promedio) = 1

/*Trae los 10 alumnos con mejor promedio*/
SELECT TOP 10 * FROM Promedios_alumnos ORDER BY Promedio DESC


CREATE VIEW AlumnosPorProvincia
AS
SELECT Provincia.Nombre AS 'Provincia', COUNT(Alumno.Dni) AS 'Cantidad de
Alumnos' FROM Alumno
       JOIN Provincia ON Provincia.Cod_provincia = Alumno.Cod_provincia
       GROUP BY Provincia.Nombre
```

### Permisos

Los permisos de Motor de base de datos se administran en el nivel de servidor mediante inicios de sesión y roles de servidor, y en el nivel de base de datos mediante usuarios de base de datos y roles base de datos. El modelo de SQL Database expone el mismo sistema dentro de cada base de datos, pero los permisos de nivel de servidor no están disponibles.

El número total de permisos para SQL Server 2019 (15.x) es 248. Azure SQL Database expone 254 permisos. La mayoría de los permisos se aplica a todas las plataformas, pero otros no. Por ejemplo, los permisos de nivel de servidor no se pueden conceder en Azure SQL Database y algunos permisos solo tienen sentido en Azure SQL Database. Los nuevos permisos se introducen gradualmente con nuevas versiones.

## Índices agrupados y no agrupados. Índices filtrados

### Índice Agrupado

es similar a una guía telefónica, los registros con el mismo valor de campo se agrupan juntos. Un índice agrupado determina la secuencia de almacenamiento de los registros en una tabla. Se utilizan para campos por los que se realizan búsquedas con frecuencia o se accede siguiendo un orden.

Una tabla sólo puede tener UN índice agrupado.

El tamaño medio de un índice agrupado es aproximadamente el 5% del tamaño de la tabla.

### Índice No Agrupado

Es como el índice de un libro, los datos se almacenan en un lugar diferente al del índice, los punteros indican el lugar de almacenamiento de los elementos indizados en la tabla. Un índice no agrupado se emplea cuando se realizan distintos tipos de búsquedas frecuentemente, con campos en los que los datos son únicos.

- Una tabla puede tener hasta 249 índices no agrupados.
- Si no se especifica un tipo de índice, de modo predeterminado será no agrupado.
- Los campos de tipo text, ntext e image no se pueden indizar.
- Es recomendable crear los índices agrupados antes que los no agrupados, porque los primeros modifican el orden físico de los registros, ordenándolos secuencialmente

### Diferencias

La diferencia básica entre índices agrupados y no agrupados es que los registros de un índice agrupado están ordenados y almacenados de forma secuencial en función de su clave.

Resumiendo, los índices facilitan la recuperación de datos, permitiendo el acceso directo y acelerando las búsquedas, consultas y otras operaciones que optimizan el rendimiento general.

## BackUp y restauración

Realizar copias de seguridad de nuestras bases de datos nos permite poder tener respaldo ante posibles anomalías en nuestros proyectos, estás pueden suceder por cause del personal humano, es decir, errores cometidos por personas que administran las bases de datos, o bien por errores en el software y/o hardware del equipo donde se aloja la base de datos. Ante un caso de estos, es esencial tener un respaldo al cual acudir de manera inmediata, por esto, es

de mucha importancia el hecho de tener copias de seguridad. A continuación, detallamos como realizar BackUp desde el motor de base de datos MS SQL Server.

También, el uso de las copias de seguridad de archivos puede aumentar la velocidad de recuperación ya que se pueden restaurar solo los archivos dañados sin tener que restaurar el resto de la base de datos.

Por ejemplo, si una base de datos está compuesta por varios archivos ubicados en diferentes discos y se producen errores en uno de ellos, solo debe restaurar el archivo situado en el disco en que se produjeron los errores. El archivo dañado se puede restaurar rápidamente, y la recuperación es más rápida que para una base de datos completa.

### Backup automatizado

En nuestro proyecto decidimos realizar backups automatizados, es decir, que periódicamente se realicen copias de seguridad.

Consideramos conveniente realizar dos tipos de copias, una semanalmente que es de tipo FULL que, como su propio nombre indica, este tipo de respaldo copia la totalidad de los datos y una diaria que es de tipo DIFERENCIAL que consiste en respaldar únicamente los datos modificados o de nueva creación, partiendo de una copia de seguridad completa anterior.

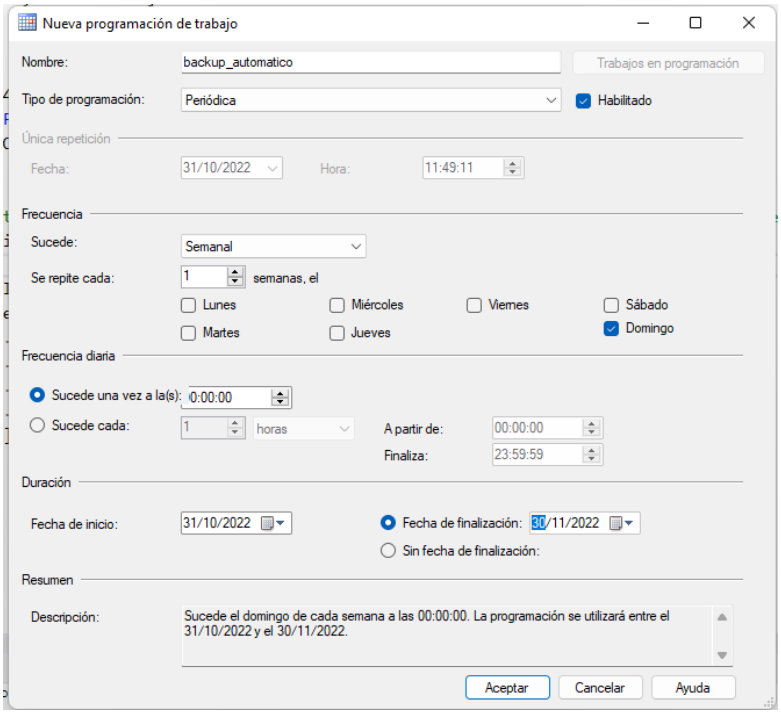
### Backup full

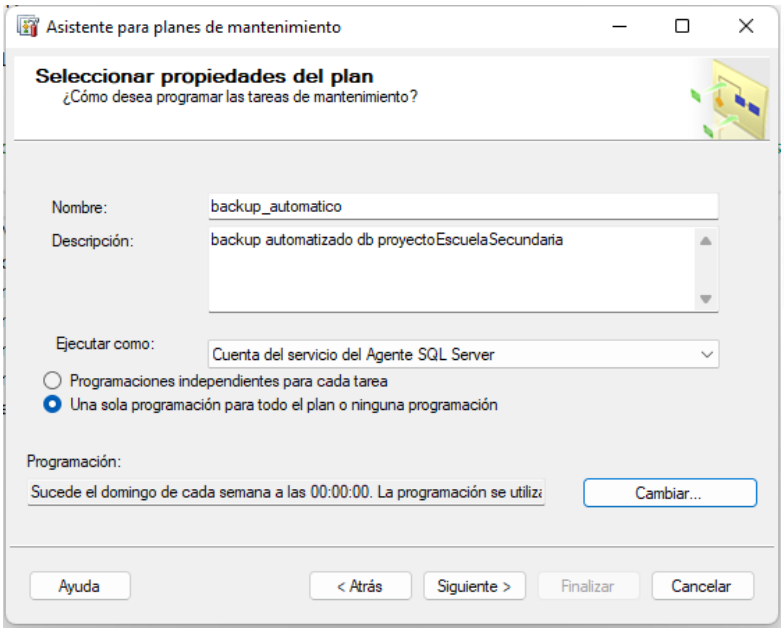
Para poder realizar la copia de tipo full, hicimos los siguientes pasos demostrados mediante recortes de pantalla.

**Paso 1:**
Primero indicamos el nombre que llevará el archivo una vez realizado la copia, seleccionamos el tipo de programación.

Seguido indicamos la frecuencia en que se va a realizar la copia indicando día y hora de la semana.

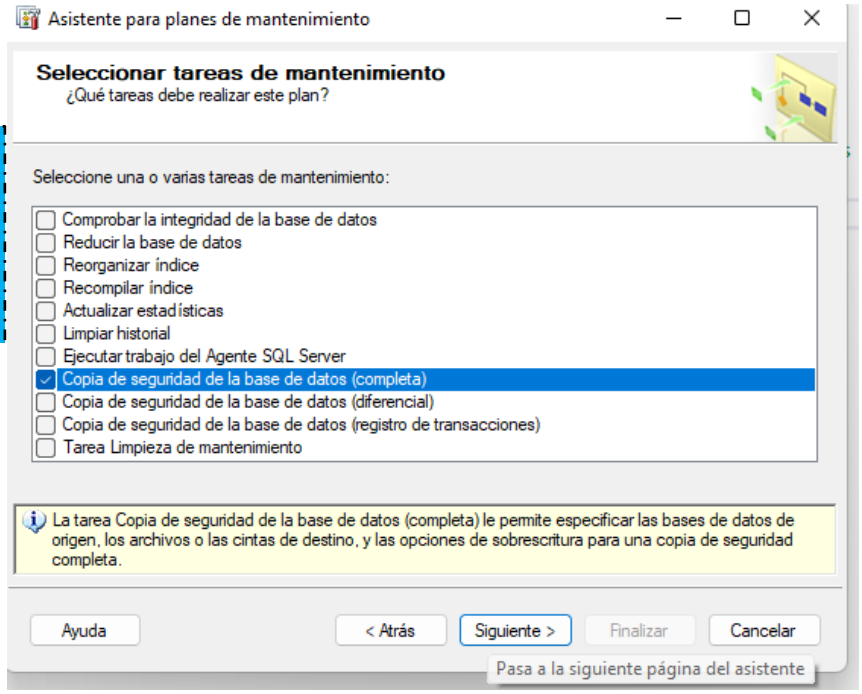
Por último, indicamos que las copias solo se van a realizar en el periodo de 31/10/2022 al 30/11/2022

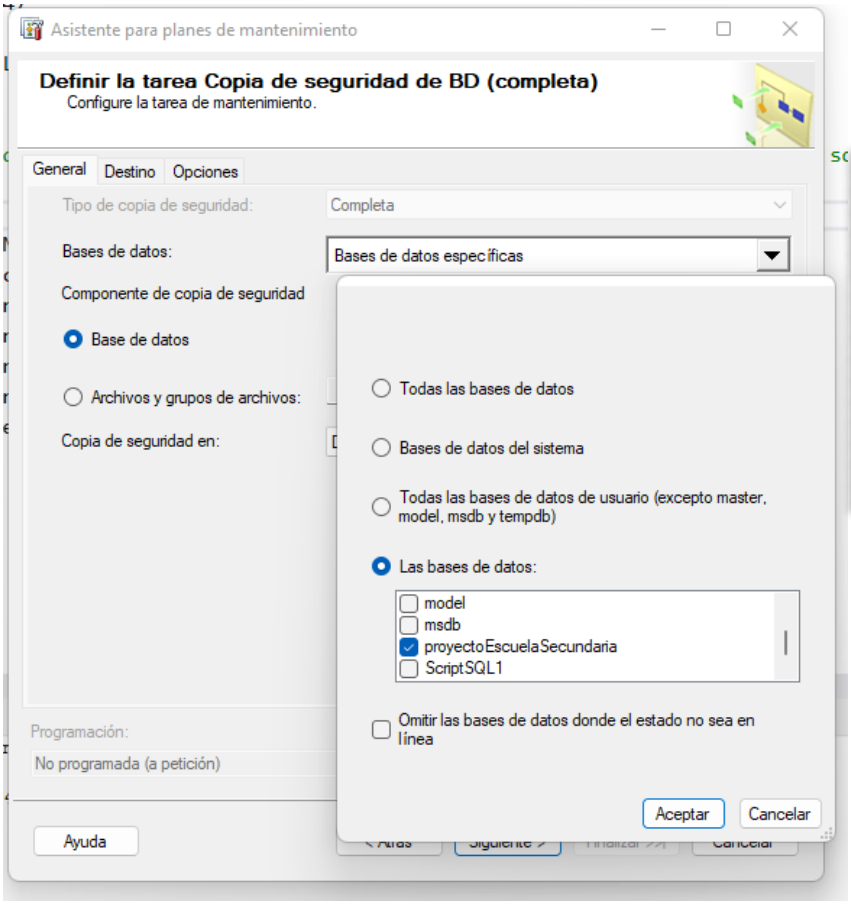





**Paso 2:**
**Seguimos con el proceso mediante el botón <Siguiente>**

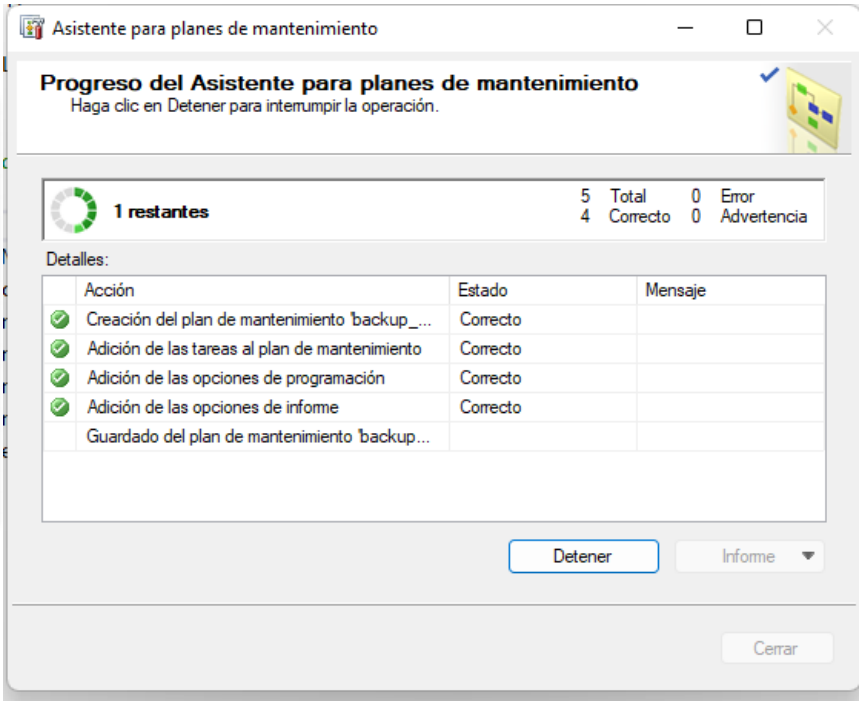
**Paso 3:**
**Indicamos el tipo de copia que se va a realizar**






**Paso 4:**

Seleccionamos a que base de datos vamos a realizar este procedimiento de backup




**Paso 5:**

Una vez terminados los pasos anteriores, solo resta esperar que se realicen las configuraciones indicadas y presionar el botón de <Aceptar>

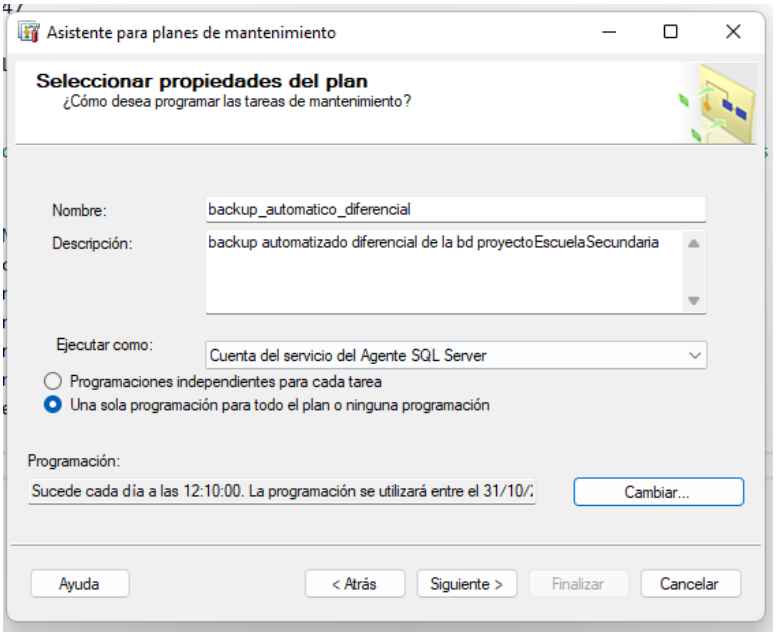
## Backup Diferencial

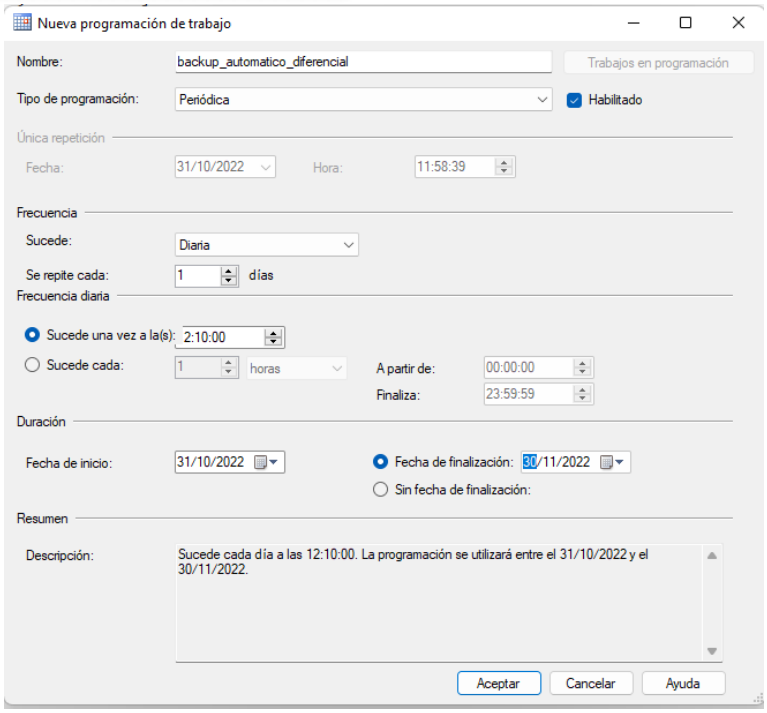
Para poder realizar la copia de tipo diferencial, hicimos los siguientes pasos demostrados mediante recortes de pantalla

**Paso 1:**
Primero indicamos el nombre que llevará el archivo una vez realizado la copia, seleccionamos el tipo de programación.

Seguido indicamos la frecuencia en que se va a realizar la copia indicando día y hora de la semana.

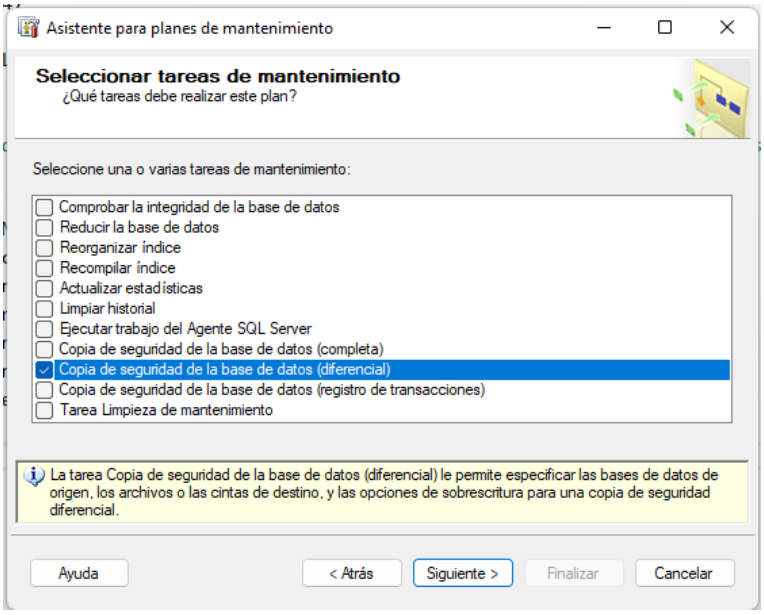
Por último, indicamos que las copias solo se van a realizar en el periodo de 31/10/2022 al

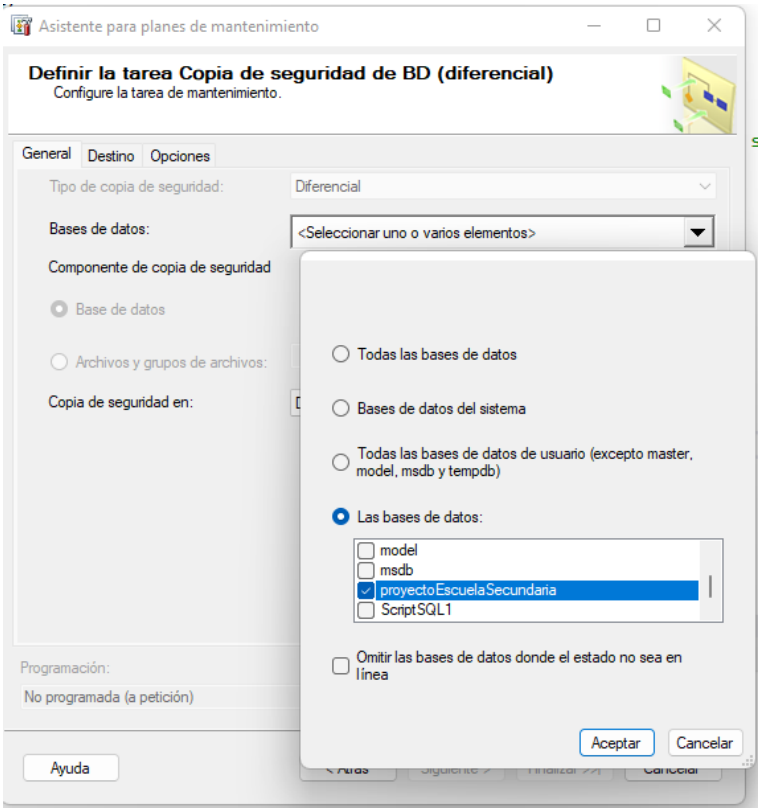



**Paso 2:**
Seguimos con el proceso mediante el botón <Aceptar>

Paso 3:
Indicamos el tipo de copia que se va a realizar



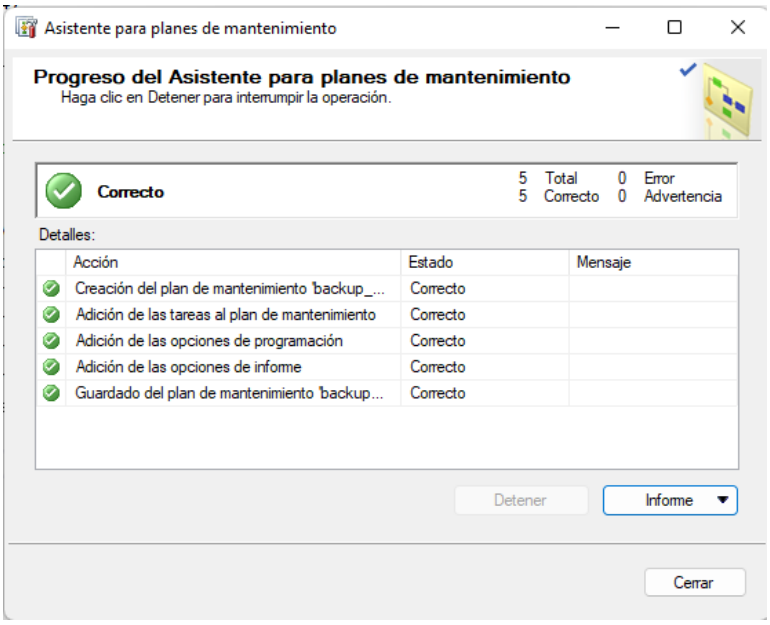




Paso 4:

Seleccionamos a que base de datos vamos a realizar este procedimiento de backup

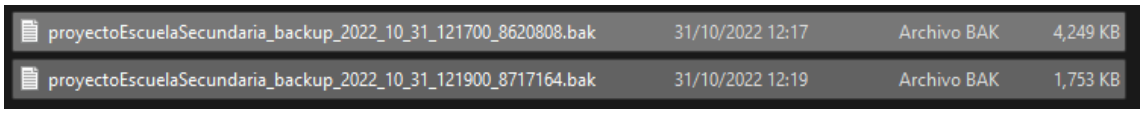
**Paso 5:**

Una vez terminados los pasos anteriores, solo resta esperar que se realicen las configuraciones indicadas y presionar el botón de <Aceptar>



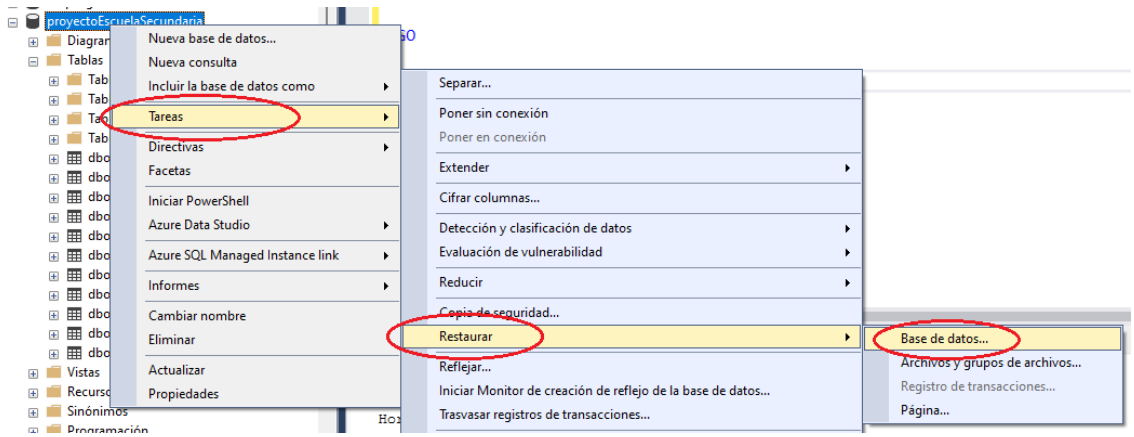

Luego de realizar estos pasos para realizar ambos tipos de backup, el archivo (llegado fecha y hora indicada en el paso 1 de cada proceso) se almacenará en la ruta C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\Backup

A continuación, adjuntamos captura de como quedan los archivos almacenados en disco C:

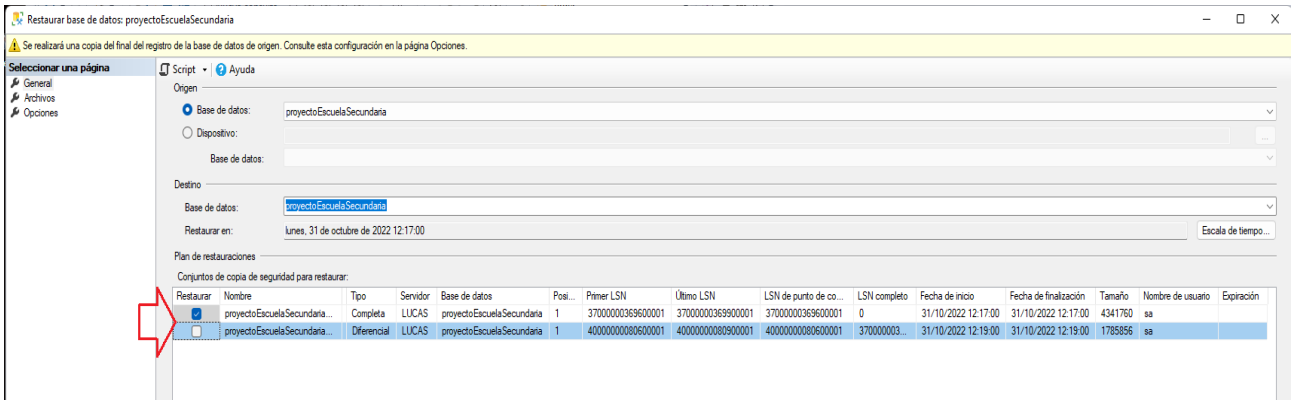



## Restauración de Backups

Ante anomalías en la base de datos, es necesario poder realizar restauración de los backups realizados con anterioridad. Para esto, a continuación, explicamos con recortes de pantallas, como restaurarlos.




Luego seleccionamos el archivo que deseamos restaurar


Restaurar base de datos: proyectoEscuelaSecundaria
Se realizará una copia del final del registro de la base de datos de origen. Consulte esta configuración en la página Opciones.
Seleccionar una página
General
Archivos
Opciones
Script
Ayuda
Origen
Base de datos:
proyectoEscuelaSecundaria
Dispositivo:
Base de datos:
Destino
Base de datos:
proyectoEscuelaSecundaria
Restaurar en:
lunes, 31 de octubre de 2022 12:17:00
Escala de tiempo...
Plan de restauraciones
Conjuntos de copia de seguridad para restaurar:
Restaurar
Nombre
Tipo
Servidor
Base de datos
Posi...
Primer LSN
Último LSN
LSN de punto de co...
LSN completo
Fecha de inicio
Fecha de finalización
Tamaño
Nombre de usuario
Expiración
proyectoEscuelaSecundaria...
Completa
LUCAS
proyectoEscuelaSecundaria
1
37000000369600001
37000000369900001
37000000369600001
0
31/10/2022 12:17:00
31/10/2022 12:17:00
4341760
sa
proyectoEscuelaSecundaria...
Diferencial
LUCAS
proyectoEscuelaSecundaria
1
40000000080600001
40000000080900001
40000000080600001
370000003...
31/10/2022 12:19:00
31/10/2022 12:19:00
1785856
sa

# Capítulo 3

## Herramientas utilizadas

Las herramientas que utilizamos para la realización del proyecto fueron las detalladas a continuación

### Discord

Fue con la que realizamos videoconferencias para la distribución de tareas e informarnos de cómo íbamos yendo en cuanto a las actividades pendientes, para estas actividades realizadas en esta aplicación creamos un nuevo servidor en el cual incorporamos chats de voz y de texto, los de voz para las videoconferencias y los de texto para ir almacenando información que creíamos importante para el desarrollo del proyecto.

### WhatsApp

Fue la aplicación de mensajería para una comunicación más rápida para cuando no nos podíamos reunir todos los integrantes

### SQL Server Management Studio

Esta herramienta decidimos utilizar para realizar el desarrollo de la base de datos, ya que es una herramienta muy completa, conocida y encontramos muy buenas referencias de la misma en Internet.

Algunas de las ventajas que nos interesaron al momento de decidir SQL Server Management Studio como gestor de base de datos utilizar fueron:

- Contiene la facilidad de soporte de transacciones
- Contiene escalabilidad estabilidad y seguridad
- También puede Soportar procedimientos almacenados
- Puede Incluir también un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- También permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y las terminales o clientes de la red sólo acceden a la información
- Además, permite administrar información de otros servidores de datos.
- También Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los terminales o clientes de la red sólo acceden a la información.
- Contiene también una base de datos relacional por lo cual también se puede incluir en la licencia de base de base datos multidimensional
- Es un producto muy maduro con mucha base implantada y muy buena comunidad.

### GitHub

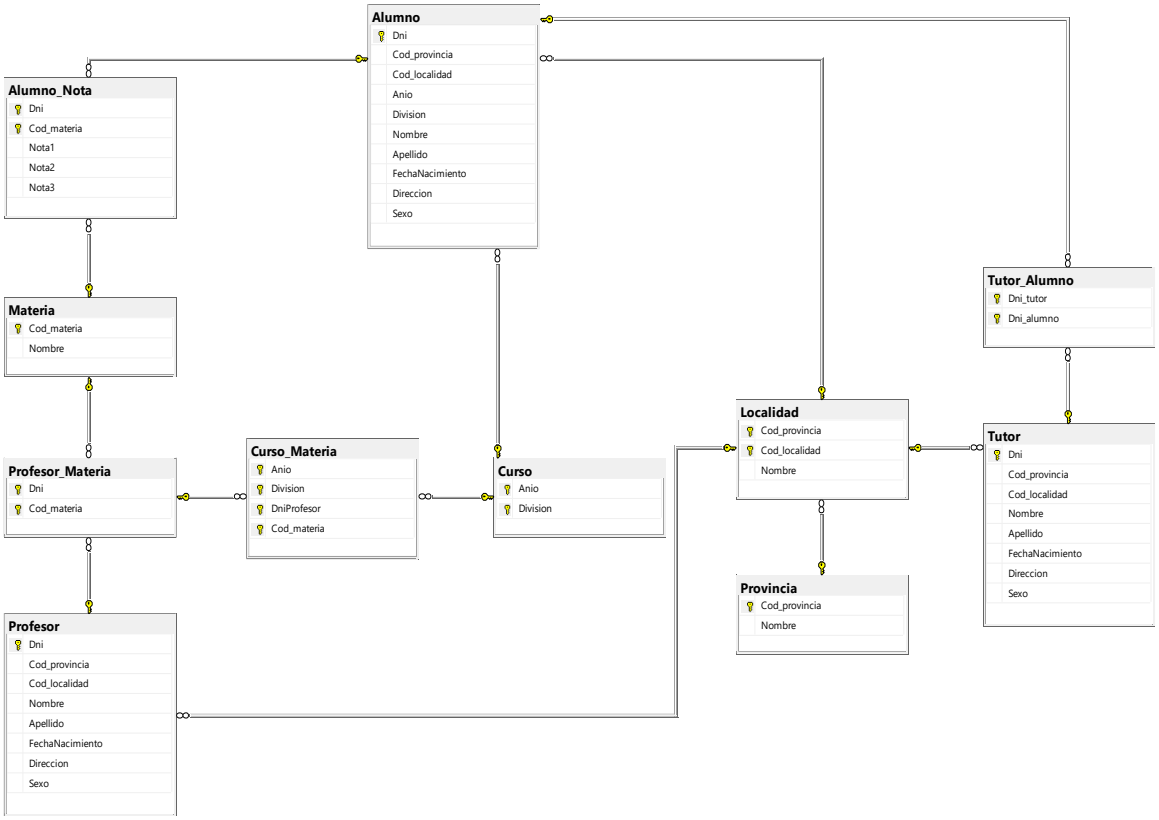
Esta herramienta colaborativa nos permitió poder trabajar en conjunto. Además, Git nos ofrece varias ventajas frente a otros sistemas tradicionales:

- Sistema distribuido, sin un punto central de fallo, que permite el trabajo incluso sin conexión.
- Superrápido y ligero, optimizado para hacer operaciones de control muy rápidas.
- Crear ramas y mezclarlas es rápido y poco propenso a problemas, al contrario que en otros sistemas tradicionales.
- La integridad de la información está asegurada gracias a su modelo de almacenamiento, que permite predecir este tipo de problemas. En sistemas tradicionales este era un problema grave.
- Permite flujos de trabajo muy flexibles.

- El concepto de área de preparación o staging permite versionar los cambios como nos convenga, no todo o nada.
- Es gratis y de código abierto.
- Otra ventaja importante es que, en la actualidad, la mayoría de empresas al momento de tomar nuevas personas en su equipo de trabajo, solicitan que tengan conocimientos en Git, lo cual nos motivó a implementarlo en nuestro proyecto para ir adquiriendo conocimientos en esta poderosa herramienta colaborativa

# Capítulo 4

A continuación, presentamos el Diagrama Entidad Relación del caso




## Diccionario de datos

### Descripción de las diferentes entidades y sus relaciones

Alumno: esta tabla contendrá los datos personales y académicos del alumno, la misma se relaciona con las tablas Alumno_Nota, Curso, Localidad, Tutor_Alumno.

Alumno_Nota: esta tabla pivote hace la conexión de mucho a muchos entre las tablas Alumno y Materia, la misma contendrá las notas del alumno en las diferentes materias

Materia: esta tabla posee código y nombre de la materia, la misma se relaciona con las tablas Alumno_Nota y Profesor_Materia

Profesor: esta tabla contiende datos personales del profesor, se relaciona con las tablas Profesor_Materia y Localidad

Profesor_Materia: esta tabla pivote realiza la conexión de mucho a muchos entre las tablas Profesor y Materia

Curso: esta tabla contendrá datos del curso, la misma tiene relación con las tablas Curso_Materia y Alumno

Curso_Materia: esta tabla pivote hace la conexión de mucho a muchos entre las tablas Curso y Materia

Localidad: esta tabla contiene descripción de una determinada localidad, tiene relación con las tablas Provincia, Profesor, Alumno y Tutor

Provincia: esta tabla contiene descripción de una determinada provincia, tiene relación con la tabla Localidad

Tutor: esta tabla contiene datos personales del tutor, se relaciona con las tablas de Localidad y Tutor_Alumno

Tutor_Alumno: esta tabla pivote contiene la relación de mucho a muchos entre las tablas Tutor y Alumno

## Atributos de las Tablas

**Alumno**

| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
|---|---|---|---|
| Dni | INT | No | Dni del alumno PK |
| Cod_provincia | INT | No | Código de provincia FK |
| Cod_localidad | INT | No | Código de localidad FK |
| Anio | INT | Si | Año del curso FK |
| Division | CHAR (1) | Si | (1 – 2 – 3 etc.) o (A – B – C etc.) FK |
| Nombre | VARCHAR (50) | No | Nombre/s del alumno |
| Apellido | VARCHAR (50) | No | Apellido/s del alumno |
| FechaNacimiento | DATE | No | Fecha nacimiento del alumno |
| Direccion | VARCHAR (255) | No | Calle y Altura de dirección |
| Sexo | CHAR (1) | No | M (Masculino) – F (Femenino) |

**Alumno_Nota**

| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
|---|---|---|---|
| Dni | INT | No | Dni del alumno PK FK |
| Cod_materia | INT | No | Código de materia |
| Nota1 | DECIMAL (4, 2) | Si | Valor decimal de la nota |
| Nota2 | DECIMAL (4, 2) | Si | Valor decimal de la nota |
| Nota3 | DECIMAL (4, 2) | Si | Valor decimal de la nota |

**Materia**

| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
|---|---|---|---|
| Cod_materia | INT | No | Código de la materia PK |
| Nombre | VARCHAR (130) | No | Nombre de la materia |

**Profesor**

| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
|---|---|---|---|
| Dni | INT | No | Dni del profesor PK |
| Cod_provincia | INT | No | Código de provincia FK |
| Cod_localidad | INT | No | Código de localidad FK |
| Nombre | VARCHAR (50) | No | Nombre del profesor |
| Apellido | VARCHAR (50) | No | Apellido del profesor |
| FechaNacimiento | DATE | No | Facha nacimiento del profesor |
| Direccion | VARCHAR (50) | No | Calle y Altura de dirección |
| Sexo | CHAR (1) | No | M (Masculino) – F (Femenino) |

| Profesor_Materia | | | |
|---|---|---|---|
| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
| Dni | INT | No | Dni del profesor PK FK |
| Cod_materia | INT | No | Código de la materia PK FK |

| Curso | | | |
|---|---|---|---|
| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
| Anio | INT | No | Año del curso PK |
| Division | CHAR (1) | No | (1 – 2 – 3 etc.) o (A – B – C etc.) PK |

| Curso_Materia | | | |
|---|---|---|---|
| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
| Anio | INT | No | Año del curso PK FK |
| Division | CHAR (1) | No | (1 – 2 – 3 etc.) o (A – B – C etc.) PK FK |
| DniProfesor | INT | No | Dni del profesor PK FK |
| Cod_materia | INT | No | Código de la materia PK FK |

| Tutor | | | |
|---|---|---|---|
| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
| Dni | INT | No | Dni del tutor PK |
| Cod_provincia | INT | No | Código de provincia FK |
| Cod_localidad | INT | No | Código de localidad FK |
| Nombre | VARCHAR (50) | No | Nombre/s del tutor |
| Apellido | VARCHAR (50) | No | Apellido/s del tutor |
| FechaNacimiento | DATE | No | Fecha nacimiento del tutor |
| Direccion | VARCHAR (255) | No | Calle y Altura de dirección |
| Sexo | CHAR (1) | No | M (Masculino) – F (Femenino) |

| Tutor_Alumno | | | |
|---|---|---|---|
| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
| Dni_tutor | INT | No | Dni del tutor PK FK |
| Dni_alumno | INT | No | Dni del alumno PK FK |

| Provincia | | | |
|---|---|---|---|
| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
| Cod_provincia | INT | No | Código de provincia PK |
| Nombre | VARCHAR (50) | No | Nombre de la provincia |

| Localidad | | | |
|---|---|---|---|
| Nombre | Tipo(longitud) | Acepta Valores NULL | Descripción |
| Cod_provincia | INT | No | Código de provincia PK FK |
| Cod_localidad | INT | No | Código de localidad PK |
| Nombre | VARCHAR (50) | No | Nombre de la localidad |

## Restricciones

| Nombre Entidad | Tipo de Restricción | Nombre de Restricción | Columna |
|---|---|---|---|
| Alumno | PRIMARY KEY | PK_Alumno | Dni |
| Alumno | FOREIGN KEY | FK_alumno_curso | Anio, Division |
| Alumno | FOREIGN KEY | FK_alumno_localidad_provincia | Cod_provincia, Cod_localidad |
| Alumno | CHECK | CK_Sexo_Alumno | Sexo |
| Alumno_Nota | PRIMARY KEY FOREIGN KEY | PK_alumno_nota | Dni, Cod_materia |
| Alumno_Nota | FOREIGN KEY | FK_alumno_alumno | Dni |
| Alumno_Nota | FOREIGN KEY | FK_alumno_materia | Cod_materia |
| Curso | PRIMARY KEY | PK_curso | Anio, Division |
| Curso_Materia | PRIMARY KEY | PK_curso_materia | Anio, Division, DniProfesor, Cod_materia |
| Curso_Materia | FOREIGN KEY | FK_curso_materia_curso | Anio, Division |
| Curso_Materia | FOREIGN KEY | FP_curso_materia_profesor_materia | DniProfesor, Cod_materia |
| Localidad | PRIMARY KEY | PK_localidad | Cod_provincia, Cod_localidad |
| Localidad | FOREIGN KEY | FK_cod_provincia | Cod_provincia |
| Materia | PRIMARY KEY | PK_Materia | Cod_materia |
| Profesor | PRIMARY KEY | PK_Profesor | Dni |
| Profesor | FOREIGN KEY | FK_profesor_localidad_provincia | Cod_provincia, Cod_localidad |
| Profesor | CHECK | CK_Sexo_Profesor | Sexo |
| Profesor_Materia | PRIMARY KEY | PK_profesor_materia | Dni, Cod_materia |
| Profesor_Materia | FOREIGN KEY | FK_profesor_materia_cod_materia | Cod_materia |
| Profesor_Materia | FOREIGN KEY | FK_profesor_materia_dni | Dni |
| Provincia | PRIMARY KEY | PK_Provincia | Cod_provincia |
| Tutor | PRIMARY KEY | PK_Tutor | Dni, Cod_provincia, Cod_localidad |
| Tutor | FOREIGN KEY | FK_tutor_localidad_provincia | Cod_provincia, Cod_localidad |
| Tutor | CHECK | CK_Sexo_Tutor | Sexo |
| Tutor_Alumno | PRIMARY KEY | PK_tutor_alumno | Dni_tutor, Dni_alumno |
| Tutor_Alumno | FOREIGN KEY | FK_tutor_alumno_alumno | Dni_alumno |
| Tutor_Alumno | FOREIGN KEY | FK_tutor_alumno_tutor | Dni_tutor |

## Script para crear la base de datos

```
-- Creación de la BDD
DROP DATABASE IF EXISTS proyectoEscuelaSecundaria
CREATE DATABASE proyectoEscuelaSecundaria;
go
-- Se seecciona la base de datos
USE proyectoEscuelaSecundaria;
--USE Agencia
go

DROP TABLE  IF EXISTS Alumno_Nota
DROP TABLE  IF EXISTS Tutor_Alumno
DROP TABLE  IF EXISTS Alumno
DROP TABLE  IF EXISTS Tutor
DROP TABLE  IF EXISTS Curso_Materia
DROP TABLE  IF EXISTS Profesor_Materia
DROP TABLE  IF EXISTS Profesor
DROP TABLE  IF EXISTS Materia
DROP TABLE  IF EXISTS Curso
DROP TABLE  IF EXISTS Localidad
DROP TABLE  IF EXISTS Provincia

DROP PROCEDURE IF EXISTS AgregarAlumno
DROP PROCEDURE IF EXISTS AgregarMateriaCurso
DROP PROCEDURE IF EXISTS EliminarMateriaCurso

-- CREAR las tablas dentro del esquema estructura
CREATE TABLE Provincia (
	Cod_provincia int PRIMARY KEY,
	Nombre varchar(50) NOT NULL,
);
CREATE TABLE Localidad (
	Cod_provincia int,
	Cod_localidad int,
	Nombre varchar(50) NOT NULL,
	CONSTRAINT PK_localidad PRIMARY KEY (cod_provincia, cod_localidad),
	CONSTRAINT FK_cod_provincia FOREIGN KEY (cod_provincia) REFERENCES
provincia(cod_provincia),
);
CREATE TABLE Tutor (
	Dni int PRIMARY KEY,
	Cod_provincia int NOT NULL,
	Cod_localidad int NOT NULL,
	Nombre varchar(50) NOT NULL,
	Apellido varchar(50) NOT NULL,
	FechaNacimiento date NOT NULL,
	Direccion varchar(255) NOT NULL,
	Sexo char(1) NOT NULL,
	CONSTRAINT CK_Sexo_Tutor CHECK(sexo IN ('M', 'F')),
	CONSTRAINT FK_tutor_localidad_provincia FOREIGN KEY (cod_provincia,
cod_localidad) REFERENCES localidad(cod_provincia, cod_localidad),
);
CREATE TABLE Profesor (
	Dni int PRIMARY KEY,
	Cod_provincia int NOT NULL,
	Cod_localidad int NOT NULL,
	Nombre varchar(50) NOT NULL,
	Apellido varchar(50) NOT NULL,
	FechaNacimiento date NOT NULL,
	Direccion varchar(255) NOT NULL,
```

```
	Sexo char(1) NOT NULL,
	CONSTRAINT CK_Sexo_Profesor CHECK(sexo IN ('M', 'F')),
	CONSTRAINT FK_profesor_localidad_provincia FOREIGN KEY (cod_provincia,
cod_localidad) REFERENCES localidad(cod_provincia, cod_localidad),
);
CREATE TABLE Materia (
	Cod_materia int PRIMARY KEY,
	Nombre varchar(130) NOT NULL,
);
CREATE TABLE Profesor_Materia (
	Dni int,
	Cod_materia int,
	CONSTRAINT PK_profesor_materia PRIMARY KEY (dni, cod_materia),
	CONSTRAINT FK_profesor_materia_dni FOREIGN KEY (dni) REFERENCES
profesor(dni),
	CONSTRAINT FK_profesor_materia_cod_materia FOREIGN KEY (cod_materia)
REFERENCES materia(cod_materia),
);
CREATE TABLE Curso (
	Anio int,
	Division char,
	CONSTRAINT PK_curso PRIMARY KEY (Anio, division),
);
CREATE TABLE Curso_Materia (
	Anio int,
	Division char,
	DniProfesor int,
	Cod_materia int,
	CONSTRAINT PK_curso_materia PRIMARY KEY (Anio, division, dniProfesor,
cod_materia),
	CONSTRAINT FK_curso_materia_curso FOREIGN KEY (Anio, division) REFERENCES
curso(Anio, division),
	CONSTRAINT FK_curso_materia_profesor_materia FOREIGN KEY (dniProfesor,
cod_materia) REFERENCES profesor_materia(dni, cod_materia),
);
CREATE TABLE Alumno (
	Dni int PRIMARY KEY,
	Cod_provincia int NOT NULL,
	Cod_localidad int NOT NULL,
	Anio int,
	Division char,
	Nombre varchar(50) NOT NULL,
	Apellido varchar(50) NOT NULL,
	FechaNacimiento date NOT NULL,
	Direccion varchar(255) NOT NULL,
	Sexo char(1) NOT NULL,
	CONSTRAINT CK_Sexo_Alumno CHECK(Sexo IN ('M', 'F')),
	CONSTRAINT FK_alumno_localidad_provincia FOREIGN KEY (Cod_provincia,
Cod_localidad) REFERENCES localidad(Cod_provincia, Cod_localidad),
	CONSTRAINT FK_alumno_curso FOREIGN KEY (Anio, Division) REFERENCES
curso(Anio, Division),
);
CREATE TABLE Tutor_Alumno (
	Dni_tutor int,
	Dni_alumno int,
	CONSTRAINT PK_tutor_alumno PRIMARY KEY (dni_tutor, dni_alumno),
	CONSTRAINT FK_tutor_alumno_tutor FOREIGN KEY (dni_tutor) REFERENCES
tutor(dni),
	CONSTRAINT FK_tutor_alumno_alumno FOREIGN KEY (dni_alumno) REFERENCES
alumno(dni),
);
```

```
CREATE TABLE Alumno_Nota (
	Dni int,
	Cod_materia int,
	Nota1 decimal(4,2),
	Nota2 decimal(4,2),
	Nota3 decimal(4,2),
	CONSTRAINT PK_alumno_nota PRIMARY KEY (dni, cod_materia),
	CONSTRAINT FK_alumno_nota_alumno FOREIGN KEY (dni) REFERENCES alumno(dni),
	CONSTRAINT FK_alumno_nota_materia FOREIGN KEY (cod_materia) REFERENCES
materia(cod_materia),
);
```

# Capítulo 5

## Conclusiones

Luego de haber finalizado este proyecto a nivel grupal, dado los conocimientos nuevos adquiridos en la materia, los que fuimos adquiriendo por medio de investigaciones necesarias para poder desarrollar este trabajo y los que ya los teníamos, estamos en condiciones de afirmar que fue un trabajo completo el cual nos permitió poder desarrollar todos los objetivos que teníamos en un principio al momento que elegimos el caso de estudio.

También, adquirimos nuevos conocimientos sobre las funciones y sus utilidades en una base de datos. Del mismo modo, los procedimientos almacenados que nos brindan la posibilidad de ejecutar instrucciones que van a ejecutar acciones dentro de nuestra base de datos.

Así como con los procedimientos almacenados, también con los disparadores o trigger, que nos brindan la posibilidad de ejecutar ciertas sentencias de forma automática cuando se realizan operaciones de tipo INSERT, UPDATE, DELETE en una tabla o una vista. También, vimos y aplicamos disparadores que nos permiten realizar auditorías dentro de nuestro proyecto, es decir, disparadores que permiten auditar a una determinada base de datos dentro del motor de base de datos MS SQL Server.

Luego vimos y trabajamos con las transacciones, que nos permiten trabajar dentro de la base de datos con un conjunto de operaciones. Así como también, tuvimos la posibilidad de estudiar y aplicar las vistas, que son un conjunto de resultados de una consulta almacenada de los datos.

De la misma forma con los permisos, que permiten a un usuario determinado poder tener privilegios y restricciones respecto a otros usuarios. En nuestro proyecto, implementamos dos tipos de usuarios, el rector y el preceptor.

Por último, queríamos concluir sobre el backup de la base de datos, lo realizamos desde el Management Studio, lo cual nos permitió poder realizar las copias de seguridad en los dos tipos que creímos necesarios, es decir, de tipo Full y Diferencial. Afirmamos que es muy importante realizar este tipo de procedimientos dentro de la base de datos ya que en un caso real esto será de mucha ayuda en caso de enfrentar errores o perdidas en la información almacenada en la misma. Del mismo modo, con la restauración a partir de las copias realizadas, no es una tarea difícil de realizar y es realmente una gran herramienta que nos permite volver a un punto donde todo estaba correcto en nuestra base de datos.

En fin, podemos afirmar que la realización de este trabajo grupal nos sirvió para comprender muchos conceptos sobre base de datos antes dados, ya que en este trabajo aplicamos muchos conocimientos teóricos que venimos adquiriendo a lo largo de la carrera hasta su actualidad.

# Bibliografías

https://learn.microsoft.com/es-es/sql/t-sql/

https://www.ibm.com/docs/es/license-metric-tool?topic=database-restoring-ms-sql-server

https://www.ibm.com/docs/es/license-metric-tool?topic=database-backing-up-ms-sql-server

https://www.tutorialesprogramacionya.com/sqlserverya/index.php?inicio=1

https://codigosql.top/sql-server/