**Tarea 4: SEGURIDAD**

[**1.- Encriptación**](#_1.-_Encriptación)

[**Veracrypt**](#_Veracrypt)

[**Encriptación de procedimientos**](#_Encriptación_de_procedimientos)

[**Encriptación de datos en SSMS**](#_Encriptación_de_datos)

[**Encriptación de columnas**](#_Encriptación_de_columnas)

[**Ejemplo 2 (encriptación columnas)**](#_Ejemplo_2_(encriptación)

[**Encriptar usando frase**](#_Encriptar_usando_frase)

[**Encriptación de Backups**](#_Encriptación_de_Backups)

[**Encriptación de TDE (Transparent data encryption)**](#_Encriptación_de_TDE)

[**Hashing**](#_Hashing)

[**DDM Dynamic Data Masking**](#_DDM_Dynamic_Data)

[**Secuencias**](#_Secuencias)

[**ROW LEVEL SECURITY (RLS)**](#_ROW_LEVEL_SECURITY)

[**Otro ejemplo de RLS**](#_Otro_ejemplo_de)

[**Always Encrypted**](#_Always_Encrypted)

[**2.- Auditorías**](#_2.-_Auditorías)

[**Auditoría de servidor**](#_Auditoría_de_servidor)

[**Desde el interfaz gráfico:**](#_Desde_el_interfaz)

[**Desde el script**](#_Desde_el_script)

[**Audiroría de base de datos**](#_Audiroría_de_base)

[**3.- Ledger**](#_3.-_LEDGER)

[**Updatable Ledger Table**](#_Updatable_Ledger_Table)

[**Append-only Ledger Table**](#_Append-only_Ledger_Table)

[**Prueba detección de modificaciones**](#_Prueba_detección_de)

## [4.- SQL DATA DISCOVERY AND CLASSIFICATION IN SSMS](#cuatro4)

[**5.- Ataques contra bases de datos**](#cinco5)

# 1.- Encriptación

La **encriptación** (o mejor dicho, **cifrado**) es el proceso de convertir información en un formato ilegible para que solo personas autorizadas puedan acceder a ella. Se utiliza para proteger datos y garantizar su confidencialidad.

En nuestro caso, lo que realmente nos interesa es que nadie pueda acceder a nuestra información, ya sea desde dentro de las bases de datos o bien desde fuera, mediante acceso a las mismas o mediante acceso a la información en circulación, capturando esta con sniffers. Para ello, el propio Windows trae un sotware de encriptación llamado Bitlocker. Para este ejercicio, nosotros hemos decidido usar Veracrypt que viene a ser lo mismo pero externo y de código abierto.

### Veracrypt

VeraCrypt es un software de **cifrado de datos** que permite crear volúmenes cifrados para proteger archivos y discos completos. Es un sucesor mejorado de **TrueCrypt**, diseñado para proporcionar mayor seguridad y resistencia contra ataques de fuerza bruta. Es útil para proteger información confidencial en entornos personales y profesionales, sobre todo en el ámbito de la **ciberseguridad**, donde la privacidad y la integridad de los datos son críticas.

Nos permitirá guardar bases de datos completas o copias de seguridad de las mismas.

Lo primero que haremos será acceder a la web oficial de Veracrypt para descargar el software, como podemos ver, tenemos el software disponible para las plataformas mas usadas. Lo descargamos (en nuestro caso para Windows) y lo instalamos, no hacemos referencia a la instalación puesto que tan solo tendremos que presionar “siguiente” y dejar que la instalación finalice.

<https://www.veracrypt.fr/en/Downloads.html>

Al abrir el software nos encontramos con la siguiente ventana, tendremos que escoger una letra de unidad y luego presionar en “Create Volume”.

Se nos mostrarán tres opciones, una para crear y encriptar un archivo contenedor, otra para encriptar un driver intero o externo (disco duro entero) y otra opción para encriptar particiones enteras o sistemas operativos. Escogemos la primera opción.

Nos mostrará un warning advirtiéndonos de que vamos a crear un volumen y que si no tenemos privilegios de administrador, puede que luego no podamos acceder a dicho volumen. Presionamos en “No”.

Ahora se nos mostrarán dos opciones, la primera una encriptación normal y la segunda una encriptación doble para que dependiendo de la contraseña que se introduzca al desencriptar, se muestren unos archivos u otros. Seleccionamos la primera opción.

Ahora seleccionamos el directorio del archivo donde se va a guardar la información encriptada.

En el siguiente paso seleccionaremos el método de encriptación, en este caso dejaremos las opciones tal cual están ya que AES y 512 es mas que suficiente para dar seguridad a los archivos.

Por último, escogeremos el tamaño que le queremos dar al volumen encriptado, debemos tener en cuenta que nuestros archivos podrán crecer de tamaño, por este motivo debemos crear el volumen con algo de margen dependiendo de lo que queramos encriptar. En este caso como vamos a usar archivos pequeños, crearemos un volumen de 100Mb.

Añadimos un password, hemos puesto la que venimos usando en todas las contraseñas desde principio de curso ‘Abcd1234.’ Aunque nos mostrará un warning diciendo que la contraseña es demasiado corta para ser segura, presionamos continuar.

Ahora se encriptará el volumen, debemos tener en cuenta que cuanto mayor tamaño tenga el volumen, mas tiempo tardará en encriptarse. Debemos mover el ratón continuamente hasta que se llene la barra inferior. Una vez termine presionaremos en “Format”.

Ahora tendremos nuestro volumen encriptado en la ubicación que hemos designado, como podemos ver, se trata de un archivo sin extensión que no podremos abrir. Para abrirlo, usaremos de nuevo el software de Veracrypt. Entramos y esta vez seleccionamos el archivo y presionaremos en “mount”, nos pedirá la contraseña que hemos configurado.

Como podemos ver, se ha creado una unidad del tamaño que hemos designado, ahora bastará con soltar dentro de la unidad los archoivos que queremos encriptar y luego, desde el software de Veracrypt presionar en Unmount para desmontar y que el archivo quede encriptado.

## Encriptación de procedimientos

Encriptar un procedimiento es algo tan sencillo como añadir “WITH ENCRYPTION” al código del mismo. De esta forma, nadie podrá ver el contenido del mismo. Vamos a generar un procedimiento que al introducir una matrícula, se elimine la misma de la base de datos de nuestra autoescuela si existiese y de lo contrario muestre un error (aunque en este caso no tiene mucho sentido encriptar el procedimiento, podría interesarnos que nos e supiese, por ejemplo, el método que usamos para subir el sueldo a los empleados o la fórmula con la que un procedimiento calcula la productividad de un empleado).

CREATE PROCEDURE sp\_EliminarVehiculo

@Matricula NVARCHAR(10)

WITH ENCRYPTION -- Con encriptación

AS

BEGIN

SET NOCOUNT ON;

-- Verificar si la matrícula existe en la tabla

IF EXISTS (SELECT 1 FROM vehiculo WHERE Matricula\_vehiculo = @Matricula)

BEGIN

DELETE FROM vehiculo WHERE Matricula\_vehiculo = @Matricula;

PRINT 'El vehículo con matrícula ' + @Matricula + ' ha sido eliminado correctamente.';

END

ELSE

BEGIN

PRINT 'La matrícula introducida no existe en la base de datos.';

END

END;

Ahora, si intentamos acceder al código del mismo, comprobamos que se nos muestra un error.

## Encriptación de datos en SSMS

### Encriptación de columnas

El sentido de encriptar una columnas no es más que cifrar ciertos datos de una tabla que no queremos que se muestren a un determinado usuario o grupos de usuarios. (muy útil en la actualidad para la protección de datos)

Necesitaremos realizar los siguientes paso:

* Crear una clave “master key”
* Crear los certificados
* Crear la “Symmetric key”.

En este ejemplo vamos a usar la encriptación de columnas para que cada profesor de la autoescuela, solo pueda acceder a los datos de sus alumnos y no a los datos del resto de alumnos del centro.

Para crear la master key, usaremos:

DROP MASTER KEY

GO

CREATE master KEY encryption BY password = 'Abcd1234.'

Comprobamos que la hemos creado correctamentre:

SELECT name KeyName,

symmetric\_key\_id KeyID,

key\_length KeyLength,

algorithm\_desc KeyAlgorithm

FROM sys.symmetric\_keys;

GO

Para crear el certificado usaremos:

CREATE CERTIFICATE profe1cert AUTHORIZATION profe1

WITH subject = 'Abcd1234.', EXPIRY\_DATE = '2025-12-31';

GO

CREATE CERTIFICATE profe2cert AUTHORIZATION profe2

WITH subject = 'Abcd1234.', EXPIRY\_DATE = '2025-12-31';

GO

Comprobamos que los hemos creado:

SELECT name CertName,

certificate\_id CertID,

pvt\_key\_encryption\_type\_desc EncryptType,

issuer\_name Issuer

FROM sys.certificates;

GO

Por último, ya solo nos queda crear la clave simétrica:

CREATE SYMMETRIC KEY SK\_01

WITH ALGORITHM = AES\_256

ENCRYPTION BY CERTIFICATE profe1cert;

GO

CREATE SYMMETRIC KEY SK\_02

WITH ALGORITHM = AES\_256

ENCRYPTION BY CERTIFICATE profe2cert;

GO

Ahora vamos a realizar las pruebas, primero abriremos la clave simétrica para usarla

OPEN SYMMETRIC KEY SK\_01

DECRYPTION BY CERTIFICATE profe1cert

GO

Ahora insertamos datos en la tabla “alumnos” encriptando teléfono y dirección con profe1:

INSERT INTO [autoescuelaP].[dbo].[alumnos] (

[DNI\_cliente],

[N\_cliente],

[Apel1\_cliente],

[Apel2\_cliente],

[Dir\_cliente],

[Email\_cliente],

[Telf\_cliente],

[localidad\_cod\_localidad]

)

VALUES

(

'12345678A',

'Carlos',

'García',

'López',

EncryptByKey(Key\_GUID('SK\_01'), 'Calle Gran Vía 123, Madrid'),

'carlos.garcia@mail.com',

EncryptByKey(Key\_GUID('SK\_01'), '637345678'),

28001

),

(

'87654321B',

'Laura',

'Martínez',

'Sánchez',

EncryptByKey(Key\_GUID('SK\_01'), 'Avenida Diagonal 456, Barcelona'),

'laura.martinez@mail.com',

EncryptByKey(Key\_GUID('SK\_01'), '633456789'),

08001

),

(

'11223344C',

'Miguel',

'Fernández',

'Rodríguez',

EncryptByKey(Key\_GUID('SK\_01'), 'Plaza del Ayuntamiento 7, Valencia'),

'miguel.fernandez@mail.com',

EncryptByKey(Key\_GUID('SK\_01'), '663123456'),

46001

);

cerramos la clave:

CLOSE SYMMETRIC KEY SK\_01;

GO

Repetimos el proceso impersonando en profe2 e insertando otros 3 alumnos:

Abrimos la clave:

OPEN SYMMETRIC KEY SK\_02

DECRYPTION BY CERTIFICATE profe2cert

GO

Añadimos datos:

INSERT INTO [autoescuelaP].[dbo].[alumnos] (

[DNI\_cliente],

[N\_cliente],

[Apel1\_cliente],

[Apel2\_cliente],

[Dir\_cliente],

[Email\_cliente],

[Telf\_cliente],

[localidad\_cod\_localidad]

)

VALUES

(

'55443322L',

'Ana',

'Ruiz',

'Gómez',

EncryptByKey(Key\_GUID('SK\_02'), 'Calle Sierpes 45, Sevilla'),

'ana.ruiz@hotmail.com',

EncryptByKey(Key\_GUID('SK\_02'), '654987321'),

41001

),

(

'98765432M',

'Javier',

'Hernández',

'Díaz',

EncryptByKey(Key\_GUID('SK\_02'), 'Gran Vía 22, Bilbao'),

'javier.hernandez@gmail.com',

EncryptByKey(Key\_GUID('SK\_02'), '688112233'),

48001

),

(

'11223344N',

'Sofía',

'Jiménez',

'Moreno',

EncryptByKey(Key\_GUID('SK\_02'), 'Paseo Independencia 10, Zaragoza'),

'sofia.jimenez@yahoo.com',

EncryptByKey(Key\_GUID('SK\_02'), '676543219'),

50001

);

Cerramos clave:

CLOSE SYMMETRIC KEY SK\_02;

GO

Ahora realizamos una consulta normal y observamos que los campos aparecen encriptados:

Vamos a realizar una consulta de los datos desde profe1 y como profe2 desencriptando los datos:

Como podemos ver, los campos que el profesor que consulta no ha introducido, no es capaz de desencriptarlos y aparecen como “NULL”.

\*\*Importante recordar que para poder borrar el certificado, debemos borrar antes la clave, sino no nos dejará por estar trabajando el certificado con la clave.

### Ejemplo 2 (encriptación columnas)

Vamos a crear un sistema para registrar pagos de alumnos, los datos de la tarjeta de crédito se almacenarán encriptados y cada profesor solo podrá ver los datos de los pagos que registró

Creamos la tabla de pagos

CREATE TABLE [autoescuelaP].[dbo].[pagos] (

[ID\_pago] INT IDENTITY(1,1) PRIMARY KEY,

[DNI\_alumno] VARCHAR(9) NOT NULL,

[Monto] DECIMAL(10,2) NOT NULL,

[Fecha\_pago] DATE NOT NULL,

[Tarjeta\_credito] VARBINARY(MAX),

[Titular\_tarjeta] VARBINARY(MAX)

);

GO

Otorgamos permisos:

GRANT SELECT, INSERT ON [pagos] TO profe1, profe2;

GO

Creamos el procedimiento para añadir los pagos:

CREATE PROCEDURE SP\_RegistrarPagoEncriptado

@DNI\_alumno VARCHAR(9),

@Monto DECIMAL(10,2),

@Fecha\_pago DATE,

@Tarjeta\_credito VARCHAR(19),

@Titular\_tarjeta VARCHAR(100)

WITH EXECUTE AS CALLER -- Esto hace que se ejecute con los permisos del que lo llama.

AS

BEGIN

DECLARE @ClaveSimetrica VARCHAR(10);

-- Usa la clave según el usuario

SET @ClaveSimetrica =

CASE USER\_NAME()

WHEN 'profe1' THEN 'SK\_01'

WHEN 'profe2' THEN 'SK\_02'

END;

-- Abrimos la clave necesaria

IF @ClaveSimetrica = 'SK\_01'

BEGIN

OPEN SYMMETRIC KEY SK\_01 DECRYPTION BY CERTIFICATE profe1cert;

END

ELSE IF @ClaveSimetrica = 'SK\_02'

BEGIN

OPEN SYMMETRIC KEY SK\_02 DECRYPTION BY CERTIFICATE profe2cert;

END

-- Insertamos los datos con tarjeta y titular encriptados

INSERT INTO [autoescuelaP].[dbo].[pagos] (

[DNI\_alumno],

[Monto],

[Fecha\_pago],

[Tarjeta\_credito],

[Titular\_tarjeta]

)

VALUES (

@DNI\_alumno,

@Monto,

@Fecha\_pago,

EncryptByKey(Key\_GUID(@ClaveSimetrica), @Tarjeta\_credito),

EncryptByKey(Key\_GUID(@ClaveSimetrica), @Titular\_tarjeta)

);

-- Cerramos la clave

IF @ClaveSimetrica = 'SK\_01'

BEGIN

CLOSE SYMMETRIC KEY SK\_01;

END

ELSE IF @ClaveSimetrica = 'SK\_02'

BEGIN

CLOSE SYMMETRIC KEY SK\_02;

END

END;

\*\*Debemos otorgar permisos a los usuarios para ejecutar el procedimiento almacenado con GRANT EXECUTE ON [dbo].[SP\_RegistrarPagoEncriptado] TO profe1, profe2

Ahora insertamos pagos como profe1:

EXEC AS USER = 'profe1';

GO

EXEC SP\_RegistrarPagoEncriptado

@DNI\_alumno = '12345678A',

@Monto = 250.50,

@Fecha\_pago = '2025-03-20',

@Tarjeta\_credito = '4111-1111-1111-1111',

@Titular\_tarjeta = 'Carlos García López';

GO

Insertamos los pagos como profe2:

EXEC AS USER = 'profe2';

GO

EXEC SP\_RegistrarPagoEncriptado

@DNI\_alumno = '55443322L',

@Monto = 300.00,

@Fecha\_pago = '2025-03-21',

@Tarjeta\_credito = '5500-0000-0000-0004',

@Titular\_tarjeta = 'Ana Ruiz Gómez';

GO

Comprobamos:

### Encriptar usando frase

Para encriptar con una frase en vez de con una clave, tan solo tenemos que usar EncryptByPassPhrase, en este caso vamos a guardar la frase en una variable para luego aplicarla.

DECLARE @FraseSecreta VARCHAR(100) = 'Voy aprobar bases de datos';

UPDATE [profesor]

SET profesor\_Dir\_profe = EncryptByPassPhrase(@FraseSecreta, 'Calle Gran Vía 78, Madrid')

WHERE profesor\_DNI\_profe = '09876543Q';

Comprobación:

### Encriptación de Backups

\*\*Debemos tener en cuenta que para hacer un backup encriptado debemos hacer todo el proceso desde master y el certificado debe generarse en master también, de lo contrario nos dará un error diciendo que no encuentra el certificado.

Primero crearemos la master key si no la tenemos creada:

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Abcd1234.';

GO

Luego creamos el certificado:

CREATE CERTIFICATE [CertificadoBackup]

WITH SUBJECT = 'Certificado encriptar backups';

GO

Ahora debemos crear un backup del certificado, además de que lo recomienda mircosoft también es aconsejable puesto que si perdemos el certificado no podremos recuperar el backup que encriptemos.

BACKUP CERTIFICATE [CertificadoBackup]

TO FILE = 'C:\Backup\Certificados\CertificadoBackup.cert'

WITH PRIVATE KEY (

FILE = 'C:\Backup\Certificados\CertificadoBackup.key',

ENCRYPTION BY PASSWORD = 'Abcd1234.'

);

GO

\*\*Debemos tener creado el directorio donde le mandamos guardar el backup.

Ahora creamos el backup de la base de datos encriptado:

BACKUP DATABASE [AutoescuelaP]

TO DISK = 'C:\Backup\AutoescuelaP.bak'

WITH COMPRESSION, ENCRYPTION(ALGORITHM = AES\_256,

SERVER CERTIFICATE = [CertificadoBackup]);

GO

Ahora abrimos otra máquina virtual, en mi caso he usado una con Windows10 y SSMS20. Me he conectado con un usuario con rol Sysadmin e identicacion SQL Server (De otra manera no me dejaba recuperar el certificado).

Creamos clave master

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Abcd1234.';

GO

Restauramos el certificado:

CREATE CERTIFICATE [CertificadoBackup]

FROM FILE = 'C:\Certificados\CertificadoBackup.cert'

WITH PRIVATE KEY (

FILE = 'C:\Certificados\CertificadoBackup.key',

DECRYPTION BY PASSWORD = 'Abcd1234.' -- Contraseña del backup del certificado

);

GO

Restauramos el backup de la base de datos:

RESTORE DATABASE [AutoescuelaP]

FROM DISK = 'C:\Backup\AutoescuelaP.bak'

WITH REPLACE

GO

\*\*Debemos mover también los archivos que use la base de datos (en este caso carpeta DATA) y debemos tener en cuenta que en la nueva máquina tendremos que darle permisos al login para que tenga control total sobre la carpeta donde se va a reescribir.

### Encriptación de TDE (Transparent data encryption)

Cifra los archivos de la base de datos (.mdf, .ndf, .ldf) y los backups (.bak).

Creamos la master key:

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Abcd1234.';

GO

Creamos el certificado:

CREATE CERTIFICATE [CertificadoTDE]

WITH SUBJECT = 'Certificado encriptar TDE';

GO

Creamos backup del certificado:

BACKUP CERTIFICATE [CertificadoTDE]

TO FILE = 'C:\Backup\Certificados\CertificadoTDE.cert'

WITH PRIVATE KEY (

FILE = 'C:\Backup\Certificados\KeyTDE.key',

ENCRYPTION BY PASSWORD = 'Abcd1234.'

);

GO

Ahora pasamos a usar nuestra base de datos, creamos la clave de encriptación:

CREATE DATABASE ENCRYPTION KEY

WITH ALGORITHM = AES\_256

ENCRYPTION BY SERVER CERTIFICATE CertificadoTDE;

GO

Y la activamos:

ALTER DATABASE [AutoescuelaP] SET ENCRYPTION ON;

GO

Ahora comprobamos el estado:

\*\* 3 significa que TDE se encuentra activado.

Ahora hacemos un backup completo y un backup de log (el de log no es imprescindible)

-- Hacemos bakcup completo

BACKUP DATABASE AutoescuelaP

TO DISK = 'C:\Backup\RecoveryfullWithTDE.bak';

GO

-- Hacemos backup de log

BACKUP LOG AutoescuelaP

TO DISK = 'C:\Backup\RecoveryWithTDE\_log.bak';

GO

En este caso, vamos a probar a restaurar la base de datos en la otra máquina virtual mediante ATTACH, para ello ejecutamos en nuestra máquina principal para detener los procesos y separar los archivos de la base de datos de la instancia:

USE master;

ALTER DATABASE AutoescuelaP SET SINGLE\_USER WITH ROLLBACK IMMEDIATE;

EXEC sp\_detach\_db 'AutoescuelaP';

GO

Ahora vamos a la nueva máquina virtual e intentamos restaurar la base de datos realizando un ATTACH pero vemos que nos da error porque falta el certificado:

Restauramos el certificado:

CREATE CERTIFICATE [CertificadoTDE]

FROM FILE = 'C:\DATA\CertificadoTDE.cert'

WITH PRIVATE KEY (

FILE = 'C:\DATA\KeyTDE.key',

DECRYPTION BY PASSWORD = 'Abcd1234.' -- Contraseña del backup del certificado

);

GO

Ahora volvemos a intentar relizar el ATTACH:

CREATE DATABASE AutoescuelaP

ON (FILENAME = 'C:\DATA\AutoescuelaP.mdf'),

(FILENAME = 'C:\DATA\AutoescuelaP\_log.ldf')

FOR ATTACH;

GO

La base de datos se crear sin errores:

### Hashing

El hashing es un proceso criptográfico que convierte una contraseña en una cadena de caracteres de longitud fija. Esto se hace usando funciones hash como SHA-2 (SHA-256, SHA-512) o MD5

El salting es una técnica adicional al hashing. Consiste en agregar una cadena aleatoria (sal) a la contraseña antes de generar el hash. Esto previene ataques como Rainbow Tables.

No puedes recuperar una contraseña almacenada en forma de **hash**, porque el **hashing es un proceso unidireccional** (no reversible). Esto significa que, una vez que una contraseña se convierte en un hash, no se puede convertir de nuevo en la contraseña original.

Aunque el hasing se suele usar para contraseñas, en este caso voy a aplicarlo a mi base de datos para ocultar los números de cuenta de los trabajadores cada vez que se inserte una nómina en la base de datos.

Para ello, vamos a seguir el ejemplo dado en clase y vamos a generar un trigger que salte cada vez que se intente insertar un registro en la tabla nominas, de esta manera el trigger cogerá los valores que se quieren insertar, seleccionará el número de cuenta, le aplicará el hash, luego se reforzará la seguridad aplicando salting y por último registrará el dato en la tabla.

Generamos el trigger:

CREATE OR ALTER TRIGGER insertar\_nomina\_hash

ON Nomina

INSTEAD OF INSERT

AS

DECLARE

@salt VARCHAR(20),@hash VARCHAR(64),

@cuenta VARCHAR(30),@hashedcuenta VARCHAR(80),

@Id\_Nomina VARCHAR(12),@Department VARCHAR(12),

@Nivel VARCHAR(12),@H\_extra VARCHAR(12),

@contrato\_ID\_contrato INT;

SELECT @Id\_Nomina = ins.Id\_Nomina,

@Department = ins.Department,

@Nivel = ins.Nivel,

@H\_extra = ins.H\_extra,

@cuenta = ins.N\_Cuenta,

@contrato\_ID\_contrato = ins.contrato\_ID\_contrato

FROM INSERTED ins;

BEGIN

SET @cuenta = (SELECT Id\_Nomina FROM inserted)

SELECT @salt = CONVERT(VARCHAR(20), CRYPT\_GEN\_RANDOM(8),2)

SET @hash = CONVERT(VARCHAR(64),HashBytes('SHA2\_256', (@salt + @cuenta)),2)

SET @hashedcuenta = @salt + @hash;

INSERT INTO Nomina ([Id\_Nomina], [Department], [Nivel], [H\_extra], [N\_Cuenta], [contrato\_ID\_contrato])

VALUES (@Id\_Nomina, @Department, @Nivel, @H\_extra, @hashedcuenta, @contrato\_ID\_contrato);

END;

GO

Ahora vamos a intentar insertar una nómina en la tabla:

INSERT INTO [dbo].[Nomina]

([Id\_Nomina], [Department], [Nivel], [H\_extra], [N\_Cuenta], [contrato\_ID\_contrato])

VALUES

('NOM011', 'IT', 'Senior', '5', 'ES1234567890123456789012', 3);

GO

Comprobamos y vemos que el número de cuenta aparece encriptado:

Aunque el hashing no se puede revertir, si que podemos aplicar el hash a una contraseña y saber si coinciden el has nuevo con el antiguo y eso nos sirve para comprobar si la contraseña introducida coincide con la guardada, ya que el hash da siempre el mismo resultado si se le aplica a lo mismo y el salt es una cadena de caracteres añadida.

Vamos a comprobarlo, aplicándolo a la cuenta a la que le hemos aplicado el salt y el hash en la tabla Nómina, de esta manera al ingresar la nomina podemos no saber que cuenta tenemos almacenada pero si podemos saber si en la que vamos hacer el ingreso es la correcta.

Generamos la función:

CREATE OR ALTER FUNCTION cuentavalida (

@Id\_Nomina varchar(20),@nuevacuenta varchar(30)

)

RETURNS bit

AS

BEGIN

DECLARE

@isValid bit,@cuentalmacenada varchar(80),@nuevacuentahash varchar(80),

@salt varchar(16),@hash varchar(64)

SET @cuentalmacenada = (SELECT [N\_Cuenta]

FROM Nomina

WHERE Id\_Nomina = @Id\_Nomina)

IF (@cuentalmacenada IS NULL) RETURN 0

SET @salt = SUBSTRING(@cuentalmacenada, 1, 16)

SET @hash = CONVERT(VARCHAR(64),HashBytes('SHA2\_256', (@salt + @nuevacuenta)),2)

SET @nuevacuentahash = @salt + @hash;

IF (@cuentalmacenada != @nuevacuentahash)

SET @isValid = 0

ELSE

SET @isValid = 1

RETURN @isValid

END;

GO

No consigo detectar el error, aunque la función no me da error, nunca me detecta el valor correcto:

Detectado el error:

El error estaba en el trigger ya que en vez de estar tomando el nº de cuenta para aplicar el hash, estaba utilizando el nº de nómina. Al cambiar la variable ya funcionó todo correctamente.

## DDM Dynamic Data Masking

Dynamic Data Masking (DDM), que es una característica de seguridad diseñada para limitar la exposición de datos sensibles a usuarios sin privilegios.

Características principales en SQL Server :  
Protección a nivel de columna: Puedes aplicar máscaras a columnas específicas que contienen información sensible  
Múltiples funciones de enmascaramiento:

- Máscara predeterminada: Muestra XXXX para tipos de datos de cadena, 0 para tipos numéricos y 01.01.1900 para fecha/hora  
- Máscara de correo electrónico: Expone solo la primera letra y el dominio (ej., a\*\*\*\*@ejemplo.com)  
- Máscara parcial: Muestra los primeros y últimos caracteres (ej., ABCD1234 → AB\*\*\*\*34)  
- Máscara de cadena personalizada: Permite patrones de reemplazo personalizados

Acceso basado en privilegios: Los usuarios con los permisos adecuados (p.ej., permiso UNMASK) pueden ver los datos originales

Creamos una tabla con campos enmascarados:

DROP TABLE IF EXISTS aprobados

GO

CREATE TABLE aprobados(

ID INT IDENTITY PRIMARY KEY,

Nombre NVARCHAR(15) MASKED WITH (FUNCTION = 'default()') NULL,

Apellidos NVARCHAR(30) MASKED WITH (FUNCTION = 'default()') NULL,

Email NVARCHAR(18) MASKED WITH (FUNCTION = 'email()') NULL,

N\_cuenta NVARCHAR (25) MASKED WITH (FUNCTION = 'partial(0,"XXXX-XXXX-XXXX-",4)') NULL

);

GO

Ahora insertamos valores:

INSERT INTO aprobados (Nombre, Apellidos, Email, N\_Cuenta)

VALUES ('Borja', 'Costa Rojo', 'cjgalego@gmail.com', '2100-2938-7765-9876-2345');

GO

Si realizamos una consulta con un usuario con permisos, se verán todos los datos pero si realizamos una consulta con un usuario sin permisos, solo con permiso de consulta, se muestra la información enmascarada de la siguiente manera:

Podemos hacer que un usuario vea los campos concediendo permisos UNMASK:

GRANT UNMASK ON aprobados TO profe1;

GO

Ahora vamos a probar a desenmascarar solo un campo, concedemos permisos a profe2 y desenmascaramos el correo:

GRANT SELECT ON aprobados TO profe2;

GO

ALTER TABLE aprobados

ALTER COLUMN email DROP MASKED;

GO

## Secuencias

Las secuencias son objetos de base de datos que generan una **secuencia numérica ordenada**, independiente de las tablas. Fueron introducidas en SQL Server 2012 y ofrecen mayor flexibilidad que las columnas de identidad. La principal diferencia con el IDENTITY es que la secuencia es un objeto independiente de la propia base de datos y podemos trabajar con la secuencia en diferentes tablas.

Vamos a suponer que queremos que a medida que registremos clientes en nuestra autoescuela, se vayan distribuyendo en aulas, en nuestra autoescuela disponemos de 3 aulas y en cada aula caben 5 alumnos. Usando una secuencia que se reinicie cada 15 alumnos (cuando se completen todas la aulas) podemos distribuir a los alumnos y calcularemos el aula que le corresponde dividiendo entre las 3 aulas. Vamos a probarlo:

-- Creamos la secuencia

CREATE SEQUENCE dbo.ControlAulas

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 15

CYCLE;

-- Actualizamos el aula automáticamente usando un valor calculado

ALTER TABLE dbo.clientes

ADD CONSTRAINT DF\_Aula DEFAULT

((NEXT VALUE FOR dbo.ControlAulas - 1)/5 % 3 + 1) FOR aula;

GO

Ahora introducimos 20 clientes y vemos si funciona:

\*\*Si tuviésemos los clientes distribuidos en varias tablas, podríamos usar igualmente la secuencia puesto que funciona en todas las tablas de la base de datos.

Podríamos por ejemplo usar esta secuencia y este supuesto para que cada 5 clientes, la secuencia diese un salto, de esta forma dejaríamos siempre un hueco vacío en cada aula para posibles incorporaciones de urgencia. Bastaría que cuando sequence valiese 5 se ejecutase un NEXT VALUE FOR [dbo].[ControlAulas].

## ROW LEVEL SECURITY (RLS)

**Row-Level Security (RLS) en SQL Server** es una funcionalidad de seguridad que permite controlar el acceso a **filas específicas** de una tabla en función de las características del usuario que ejecuta la consulta. Su objetivo es garantizar que los usuarios solo puedan ver o modificar los datos relevantes para su contexto, sin necesidad de modificar la lógica de la aplicación.

**Predicados de filtro:** Limitan las filas que se pueden leer (SELECT)  
  
**Predicados de bloqueo:** Limitan las filas donde se pueden realizar operaciones de modificación (INSERT, UPDATE, DELETE)

Vamos a intentar implementar un Row Level Security para que cada profesor de practica solo tenga acceso a las prácticas de sus alumnos y no al resto de practicas de los demás alumnos

Para ello comenzamos creando la función de seguridad:

CREATE FUNCTION dbo.fn\_seguridad\_profesor(@DNI\_profe AS SYSNAME)

RETURNS TABLE

WITH SCHEMABINDING

AS

RETURN (

SELECT 1 AS acceso

WHERE @DNI\_profe = USER\_NAME()

OR IS\_ROLEMEMBER('db\_owner') = 1

);

GO

Ahora creamos la política de seguridad:

CREATE SECURITY POLICY PoliticaSeguridadPractica

ADD FILTER PREDICATE dbo.fn\_seguridad\_profesor(profe\_practicas\_profesor\_DNI\_profe)

ON dbo.practica

WITH (STATE = ON);

GO

Ahora creamos un usuario para un profesor y hacemos una consulta impersonandonos:

Como podemos ver, solo se muestran las prácticas realizadas por el propio profesor. Sin embargo, si hacemos la misma consulta con un usuario con todos los permisos, observamos que se muestran todos los datos de todos los clientes:

Para entenderlo mejor, se podría decir que la función de seguridad es un listado de reglas sobre quien puede o no puede acceder a ciertos datos y la política de seguridad es la que nos dice a que aplicamos la función de seguridad y en que casos.

\*\*Debemos tener en cuenta que si solo usamos FILTER PREDICATE, los usuarios que no puedan ver el contenido si que podrán editarlo, eso quiere decir que si un usuario conoce algún dato de la tabla podría modificarlo o borrarlo. Para evitar este tipo de fallos siempre se debe usar FILTER PREDICATE combinado con BLOCK PREDICATE.

### Otro ejemplo de RLS

Vamos a suponer que tenemos una tabla trabajadores con sus departamentos y sus usuarios, queremos que cada usuario solo tenga permisos para consultar y editar datos de su propio departamento:

CREATE TABLE trabajadores (

id INT IDENTITY(1,1) PRIMARY KEY,

nombre VARCHAR(50) NOT NULL,

apellidos VARCHAR(100) NOT NULL,

telefono VARCHAR(15) NOT NULL,

sueldo DECIMAL(10,2) NOT NULL,

puesto VARCHAR(50) NOT NULL,

usuario VARCHAR(50) NOT NULL,

departamento VARCHAR(50) CHECK (departamento IN ('Recursos humanos', 'Logística', 'Administración')));

GO

Creamos la función de seguridad:

CREATE FUNCTION dbo.fn\_FiltroPorDepartamento(@departamento VARCHAR(50))

RETURNS TABLE

WITH SCHEMABINDING

AS

RETURN (

SELECT 1 AS acceso

FROM dbo.trabajadores

WHERE departamento = @departamento AND usuario = USER\_NAME()

);

GO

Creamos la política de seguridad:

CREATE SECURITY POLICY Trabajadores\_Policy

ADD FILTER PREDICATE dbo.fn\_FiltroPorDepartamento(departamento) ON dbo.trabajadores,

ADD BLOCK PREDICATE dbo.fn\_FiltroPorDepartamento(departamento) ON dbo.trabajadores

WITH (STATE = ON);

GO

Y ahora realizamos pruebas, el usuario de recursos humanos puede ver y editar datos de trabajadores de su departamento, incluso añadir trabajadores si son de su propio departamento pero no podrá ver ni editar nada de otros departamentos:

Ahora probamos a insertar un trabajador de otro departamento y nos da error:

Sin embargo si la inserción es de su propio departamento si nos deja:

## Always Encrypted

Encriptar la circulación de datos entre un cluente y el servidor.

**Column Encryption Key (CEK)**

La **Column Encryption Key (CEK)** es una clave utilizada para cifrar y descifrar los valores de las columnas de datos. Esta clave es específica de cada columna que deseas cifrar.

**Column Master Key (CMK)**

La **Column Master Key (CMK)** es una clave maestra que se utiliza para cifrar las Column Encryption Keys (CEK). Es la clave de más alto nivel y se asegura de proteger las CEK de manera segura.

**Cifrado Determinístico (Deterministic Encryption)**

El **cifrado determinístico** genera el mismo valor cifrado para una entrada de datos idéntica cada vez que se cifra. Esto significa que si tienes dos filas con el mismo valor en una columna, su valor cifrado será el mismo, sin importar cuántas veces se cifre.

**Cifrado Aleatorio (Randomized Encryption)**

El **cifrado aleatorio** genera un valor cifrado diferente cada vez que se cifra el mismo dato, incluso si los valores de entrada son iguales. Esto significa que, aunque los datos originales sean idénticos, su representación cifrada será diferente en cada operación de cifrado.

Vamos a la barra lateral y pinchamos sobre nuestra base de datos con el boton derecho, luego Task y luego “Encrypt Columns…”.

Ahora seleccionamos las columnas que queremos encriptar, en este caso vamos a encriptar los datos de las tarjetas de crédito usadas en los pagos, podemos escoger entre cifrado deterministico o random:

La siguiente pantalla la dejamos tal y compo está, para que se almacene en windows (podríamos almacenar en Azure también).

Una vez termine de realizar las operaciones, realizamos la consulta y comprobamos que los datos están encriptados

# 2.- Auditorías

Las auditorías en SQL Server permiten registrar eventos y actividades dentro del servidor de base de datos. Son útiles para monitorear seguridad, cumplimiento normativo y cambios en los datos.

## Auditoría de servidor

### Desde el interfaz gráfico:

* **Application log:** La captura se envía al visor de sucesos Aplications
* **Security log:** La captura va a visor de sucesos Security
* **File:** La captura va a un fichero

### Desde el script

Siempre deben realizarse desde la base de datos master.

Primero realizamos la auditoría al appLog:

USE MASTER

GO

CREATE SERVER AUDIT [Auditoria\_marzo20225]

TO application\_log

WITH

(queue\_delay = 1000,

on\_failure = fail\_operation

)

GO

Ahora la auditoría al Security log:

CREATE SERVER AUDIT [Security\_Auditoria\_marzo20225]

TO security\_log

WITH

(queue\_delay = 1000,

on\_failure = fail\_operation

)

GO

Ahora vamos a realizar la auditoría a archivo:

Primero creamos la carpeta donde se guardará:

EXEC sp\_configure 'show advanced options', 1;

RECONFIGURE;

EXEC sp\_configure 'xp\_cmdshell', 1;

RECONFIGURE;

EXEC sp\_configure 'show advanced options', 1;

RECONFIGURE;

EXEC xp\_cmdshell 'mkdir C:\auditorias';

Ahora generamos la auditoría:

CREATE SERVER AUDIT [File\_Auditoria\_marzo20225]

TO FILE

(filepath = 'c:\auditorias\',

maxsize = 0mb,

max\_rollover\_files = 2147483647,

reserve\_disk\_space = off

)

WITH(

queue\_delay = 1000,

on\_failure = continue

)

GO

Ahora tendríamos las tres auditorías creadas pero no estarían activas, para activarlas podemos hacerlo mediante el entorno gráfico, pulsando con el boton derecho sobre la auditoría o bien mediante el script:

ALTER SERVER AUDIT File\_Auditoria\_marzo20225 WITH (STATE = ON)

GO

Desaparece la X roja de la auditoría en la barra lateral.

Ahora nos queda darle las indicaciones de lo que queremos que audite, vamos a probar a hacerlo desde el script ya que desde el entorno gráfico es sencillo mediante boton derecho sobre la auditoría. Vamos a pedirle que registre inicios de sesión fallidos, inicios de sesión exitosos y cambios de contraseña.

CREATE SERVER AUDIT SPECIFICATION [Specificaciones\_Audit\_File]

FOR SERVER AUDIT [File\_Auditoria\_marzo20225]

ADD (FAILED\_LOGIN\_GROUP),

ADD (SUCCESSFUL\_LOGIN\_GROUP),

ADD (USER\_CHANGE\_PASSWORD\_GROUP)

GO

Ahora haremos varias operaciones como cambiar de usuario, cambiar contraseñas…etc y comprobamos que se registrasen los cambios en la auditoría:

\*\*No sabemos el motivo pero a pesar de que parece que todo está funcionando correctamente ya que al consultar mediante el script se muestran los registros, vemos que a través del entorno gráfico nos da un error y no se nos muestran. He mirado foros y consultado a distintas IA pero no he encontrado solución.

## Audiroría de base de datos

Viene a ser practicamente lo mismo que la auditoría de servidor pero a nivel base de datos, se usa el registro creado a nivel servidor y se implementa una especificación a nivel base de datos. En este caso hemos probado a decirle que audite operaciones de lectura, escritura y borrado.

CREATE DATABASE AUDIT SPECIFICATION [Auditoría\_Autoescuela\_2025]

FOR SERVER AUDIT [File\_auditoria\_marzo2025]

ADD (SELECT ON OBJECT::[trabajadores] BY [dbo]),

ADD (INSERT ON OBJECT::[trabajadores] BY [dbo]),

ADD (DELETE ON OBJECT::[trabajadores] BY [dbo])

GO

SELECT \* FROM trabajadores;

GO

\*\*Debemos acordarnos siempre de activar las especificaciones

ALTER DATABASE AUDIT SPECIFICATION [Auditoría\_Autoescuela\_2025] WITH (STATE = ON);

GO

# 3.- LEDGER

LEDGER en SQL Server es una extensión de las bases de datos tradicionales que proporciona un registro inmutable de los datos. Permite a los usuarios verificar que los datos no han sido alterados ni manipulados desde su creación.

**Tabla de Historial (History Table)**

Cada tabla LEDGER genera automáticamente una tabla de historial, que guarda los registros anteriores cuando se realizan actualizaciones o eliminaciones.

**Tabla de Ledger (Ledger Table)**

Es una tabla que almacena datos de forma inmutable. Existen dos tipos:

## ****Updatable Ledger Table****

Permite operaciones **INSERT, UPDATE y DELETE**, pero mantiene un historial inmutable de los cambios.

Creamos una tabla con ledger activo:

CREATE TABLE [dbo].[pagos](

[ID\_pago] [int] IDENTITY(1,1) NOT NULL,

[DNI\_alumno] [varchar](9) NOT NULL,

[Monto] [decimal](10, 2) NOT NULL,

[Fecha\_pago] [date] NOT NULL,

[Tarjeta\_credito] [varchar](25) NOT NULL,

[Titular\_tarjeta] [varchar](60) NOT NULL)

WITH

(

SYSTEM\_VERSIONING = ON (HISTORY\_TABLE = dbo.PagosHistory),

LEDGER = ON

);

GO

Comprobamos los registros del ledger:

Podemos obtener los registros del ledger a través de la vista:

Ahora realizamos varias operaciones en la tabla:

INSERT INTO [dbo].[pagos] ([DNI\_alumno], [Monto], [Fecha\_pago], [Tarjeta\_credito], [Titular\_tarjeta])

VALUES

('12345678Z', 60.00, '2024-01-15', '5555 5555 1111 1111', 'Juana Mayol');

GO

UPDATE pagos

SET Tarjeta\_credito = '5555 5555 1111 5555'

WHERE DNI\_alumno = '12345678Z'

GO

DELETE pagos

WHERE DNI\_alumno = '12345678Z';

GO

DELETE pagos

WHERE DNI\_alumno = '23456789B';

GO

Consultamos la vista y comprobamos que aparecen las operaciones realizadas:

## ****Append-only Ledger Table****

Solo permite **INSERT**, evitando cambios o eliminaciones de datos.

Creamos la tabla examen con APPEND-ONLY:

CREATE TABLE [dbo].[examen](

[ID\_examen] [int] NOT NULL,

[F\_examen] [date] NULL,

[Tipo\_examen] [nvarchar](10) NULL,

[Tipo\_carnet] [nvarchar](10) NULL,

[cliente\_DNI\_cliente] [varchar](9) NOT NULL,

[vehiculo\_vehiculo\_ID] [numeric](28, 0) NOT NULL)

WITH

(LEDGER = ON (APPEND\_ONLY = ON));

GO

Ahora vemos como podemos insertar registros en la tabla pero no nos deja borrar ni actualizar registros:

## Prueba detección de modificaciones

**sys.database\_ledger\_blocks**

Es una vista del sistema en SQL Server que almacena el historial de los bloques de auditoría (Ledger Blocks) en una base de datos con Ledger activado.  
Cada bloque representa un conjunto de transacciones registradas en las tablas ledger para garantizar la inmutabilidad y la integridad de los datos.

SELECT \* FROM sys.database\_ledger\_blocks

Go

**sp\_generate\_database\_ledger\_digest**

El Ledger Digest es un resumen criptográfico (hash) de los datos almacenados en las tablas ledger de SQL Server. Se usa para garantizar la integridad y la inmutabilidad de los datos en una base de datos con Ledger activado.

Generamos el ledger digest :

EXECUTE sp\_generate\_database\_ledger\_digest

go

Nos da como resultado:

{"database\_name":"AutoescuelaP","block\_id":0,"hash":"0xCE4736A4A458A617ACADE243D1BF35539A32FA8853A62F5F3CEC62FCA4A9089A","last\_transaction\_commit\_time":"2025-03-21T01:32:13.8433333","digest\_time":"2025-03-21T12:25:27.7360992"}

**SET ALLOW\_SNAPSHOT\_ISOLATION ON**

La Snapshot Isolation es un nivel de aislamiento en SQL Server que **evita bloqueos de lectura** y permite que las consultas lean datos consistentes sin ser afectadas por transacciones concurrentes.

ALTER DATABASE AutoescuelaP

SET ALLOW\_SNAPSHOT\_ISOLATION ON;

GO

**sp\_verify\_database\_ledger**

Verifica la integridad del historial de transacciones en la tabla dentro de la base de datos, usando los valores de ledger digest proporcionados.

EXECUTE sp\_verify\_database\_ledger N'

[{"database\_name":"AutoescuelaP","block\_id":0,"hash":"0xCE4736A4A458A617ACADE243D1BF35539A32FA8853A62F5F3CEC62FCA4A9089A","last\_transaction\_commit\_time":"2025-03-21T01:32:13.8433333","digest\_time":"2025-03-21T12:25:27.7360992"}

]';

GO

**sys.fn\_PhysLocFormatter(%%physloc%%)**

Devuelve la dirección de almacenamiento de cada fila con el formato: (File:Page:Slot)

* File: Número del archivo de datos donde se almacena la fila.
* Page: Número de la página dentro del archivo.
* Slot: Posición de la fila dentro de la página.

SELECT sys.fn\_PhysLocFormatter(%%physloc%%) as [Physical RID], \*

FROM dbo.pagos

Go

Podemos solicitar que nos devuelva una posición de una fila en concreto filtrando por el ID, en este caso vamos a pedir que nos muestre en que dirección se encuentra almacenado el registro 5, que es el que queremos modificar para ver si el ledger detecta la intrusión, por ejemplo:

SELECT sys.fn\_PhysLocFormatter(%%physloc%%) as [Physical RID], \*

FROM dbo.pagos

WHERE ID\_pago = 5;

GO

**DBCC TRACEON(3604)**

Activa una traza específica (Trace Flag 3604), que redirige la salida de ciertos comandos de depuración a la ventana de resultados en SSMS en lugar de almacenarla internamente. Cuando ejecutas comandos de diagnóstico como DBCC PAGE o DBCC IND, SQL Server no muestra los resultados por defecto. Activar TRACEON(3604) permite que se impriman en la consola de SSMS.

Vamos a pedir que se nos muestre lo que contiene la página del registro que queremos modificar:

DBCC TRACEON(3604)

GO

DBCC PAGE(AutoescuelaP, 1, 1344, 3) -- Muestra la estructura de la página 2500

GO

Ahora vamos a convertir a VARBINARY los números de las tarjetas, tanto la buena como la pirateada:

SELECT CONVERT (VARBINARY(33),'6011 0000 0000 0004') -- Conversión de tarjeta buena

GO

-- 0x36303131203030303020303030302030303034

SELECT CONVERT (VARBINARY(33),'6666 6666 6666 6666') -- Conversión de tarjeta pirateada

GO

-- 0x36363636203636363620363636362036363636

Ahora viene la parte en la que “corrompemos” los datos de la base de datos y que pretendemos detectar con el ledger. Para ello, primero vamos a desactivar la verificación de páginas:

ALTER DATABASE AutoescuelaP SET PAGE\_VERIFY NONE

GO

Cambiamos la base de datos a modo monousuario:

ALTER DATABASE AutoescuelaP SET SINGLE\_USER

GO

Ahora reescribimos lo que queremos en la página:

DBCC WRITEPAGE ('AutoescuelaP',1,544,104,33,0x36363636203636363620363636362036363636,0)

GO

Ahora volvemos a poner la base de datos en multiusuario y activamos el control de página:

ALTER DATABASE AutoescuelaP SET MULTI\_USER

GO

ALTER DATABASE AutoescuelaP SET PAGE\_VERIFY CHECKSUM

GO

Al realiar una consulta nos damos cuenta que hemos cometido un error al introducir el comando para modificar la tarjeta:

El error estaba en que habíamos introducido mal el offset de la sentencia de modificación, ahora si vemos la tarjeta modificada:

Ahora al comprobar la integridad de la base de datos con ledger, nos dice que el hash asociado no coincide con el hash actual y por tanto la base de datos está corrupta o modificada.

EXECUTE sp\_verify\_database\_ledger N'

[{"database\_name":"AutoescuelaP","block\_id":0,"hash":"0xCE4736A4A458A617ACADE243D1BF35539A32FA8853A62F5F3CEC62FCA4A9089A","last\_transaction\_commit\_time":"2025-03-21T01:32:13.8433333","digest\_time":"2025-03-21T12:25:27.7360992"}

]';

GO

## 

## 4.- SQL DATA DISCOVERY AND CLASSIFICATION IN SSMS

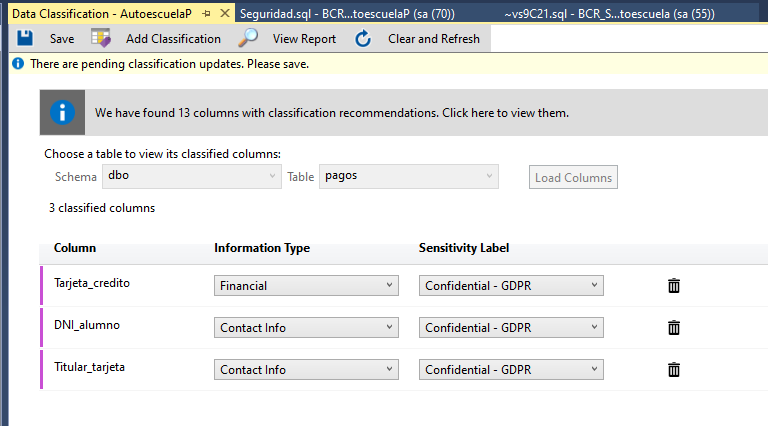
## Es una herramienta en SSMS que sirve para clasificar el tipo de información que se guarda en cada columna según la sensibilidad de la misma. En un primer momento, el propio SSMS nos realiza una primera clasificación automática que luego nosotros podremos modificar o bien añadir nuevas clasificaciones.Pinchamos con el botón derecho sobre nuestra base de datos y luego en “Task->Data Discovery and Classification->Clasify Data”.

## 

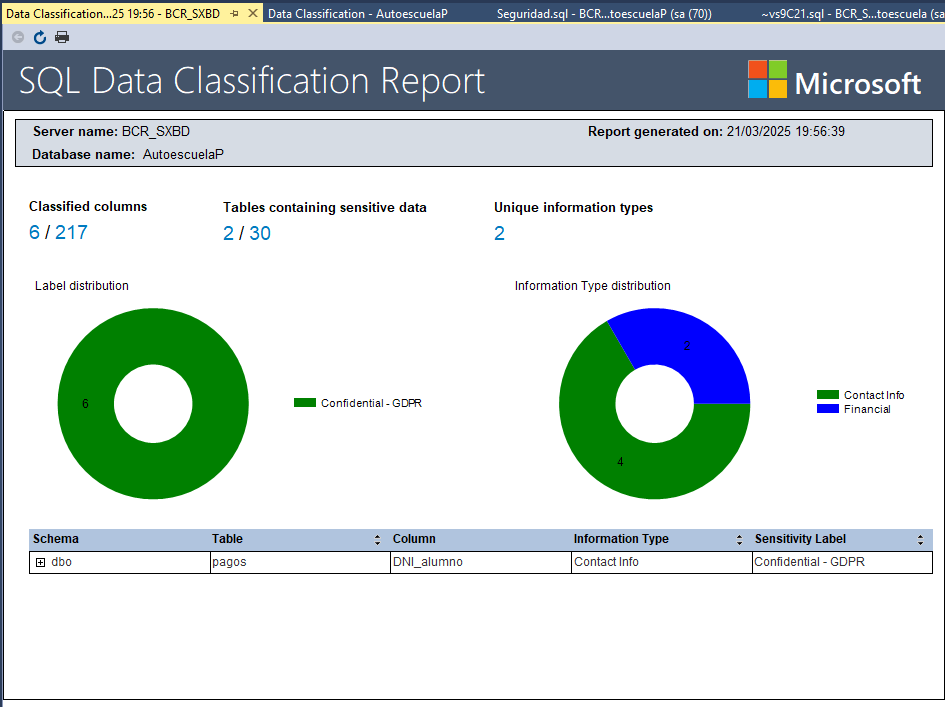
## 

## Aunque en un primer momento, el propio SSMS nos hace una clasificación, si pinchamos en la parte superior en “Add classification”, podemos añadir nosotros la clasificación que queramos de cada dato de la base de datos. Se desplegará una columna lateral en donde escogeremos la colimna, el tipo de información y la etiqueta de seguridad. Por ejemplo a la cuenta bancaria o tarjeta de crédito podemos ponerle “Finanacial” y “Confidencial GDPR”.

## 



Además, si pinchamos en “View report” se nos generará un reporte de los distintos tipos de datos y clasificación.



**5.- Ataques contra bases de datos**