“Año De La Recuperación Y Consolidación De La Economía Peruana”

UNIVERSIDAD PERUANA LOS ANDES

“FACULTAD DE INGENIERÍA”

ESCUELA PROFESIONAL “SISTEMAS Y COMPUTACIÓN”

Seguridad y Control de Acceso

CURSO: Base de Datos II

PROFESOR: Ing. Fernández Bejarano Raúl Enrique

ESTUDIANTE: Carrillo Cárdenas José

CICLO: V

SECCIÓN: A1

HUANCAYO PERÚ

2025

Semana 11: Seguridad y control de acceso

Objetivo: Implementar políticas de seguridad robustas y administración de roles. Temas:

1.Autenticación SQL y Windows: diferencias y prácticas seguras.

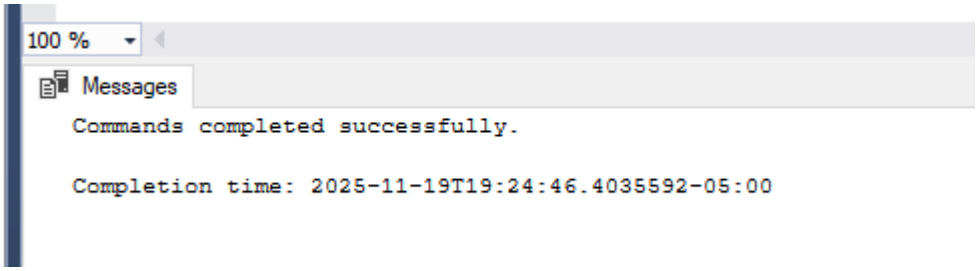
Script de la solución en T-SQL

Característica	Autenticación de SQL Server	Autenticación de Windows
Gestión de credenciales	Nombres de usuario y contraseñas creados y almacenados dentro de SQL Server.	Usa las credenciales de Windows/Active Directory del usuario para la autenticación.
Seguridad	Menos segura porque las contraseñas se almacenan y transmiten.	Más segura, ya que delega la autenticación a un sistema más robusto como Active Directory y no almacena credenciales directamente en SQL Server.
Uso de contraseñas	Se requiere un nombre de usuario y contraseña de SQL Server además del de Windows.	No requiere contraseñas de SQL Server. La validación de identidad es manejada por Windows.
Políticas de seguridad	Las políticas de contraseñas son más limitadas en comparación con las de Windows.	Permite aprovechar las políticas de contraseñas, grupos y otras funciones de seguridad de Active Directory.

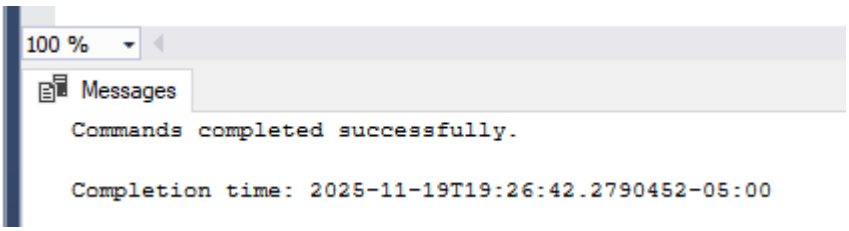
-- 1) Crear login SQL seguro

```
USE master;
GO
-- Crea un login SQL que: aplica políticas de Windows (CHECK_POLICY),
-- obliga expiración de contraseña (CHECK_EXPIRATION), y obliga cambio inicial
(MUST_CHANGE).
-- Nota: MUST_CHANGE obliga al primer inicio de sesión a cambiar la contraseña.
-- 1) Crear login SQL
IF NOT EXISTS (SELECT 1 FROM sys.server_principals WHERE name =
'login_sql_alumno')
BEGIN
    CREATE LOGIN login_sql_alumno
    WITH PASSWORD = 'P@ssw0rdSegur0!2025',
         CHECK_POLICY = ON,
```

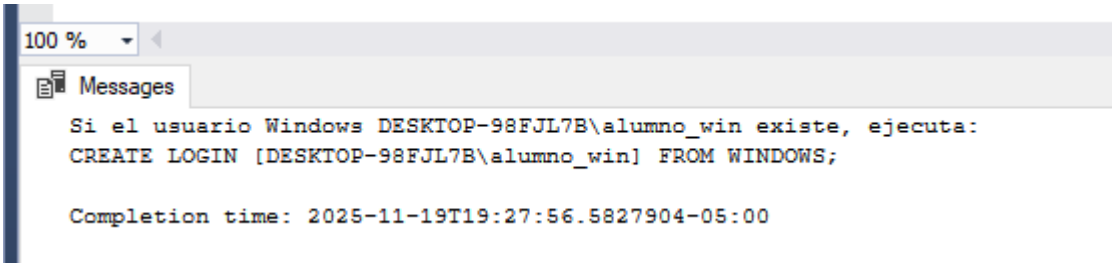
```
        CHECK_EXPIRATION = ON;
    PRINT 'Login creado sin MUST_CHANGE (requerido por SQL Server).';
END

GO
```



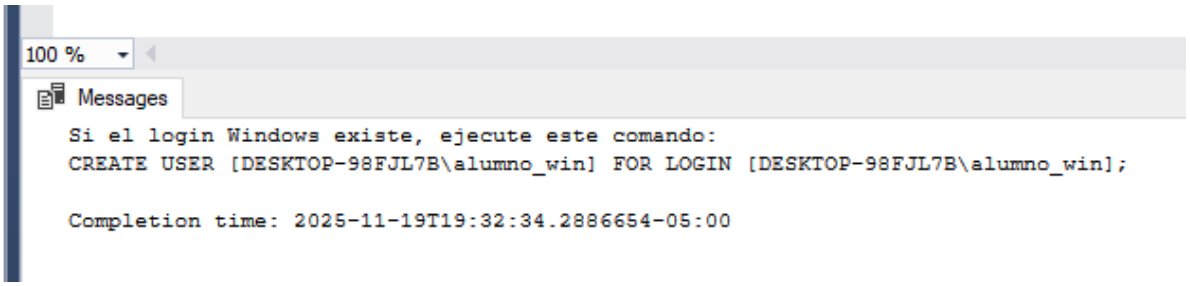

```
-- 2) Forzar cambio de contraseña en batch separado
IF NOT EXISTS (SELECT 1 FROM sys.server_principals WHERE name =
'login_sql_alumno')
BEGIN
    CREATE LOGIN login_sql_alumno
    WITH PASSWORD = 'P@ssw0rdSegur0!2025',
         CHECK_POLICY = ON,
         CHECK_EXPIRATION = ON;
END

GO
```



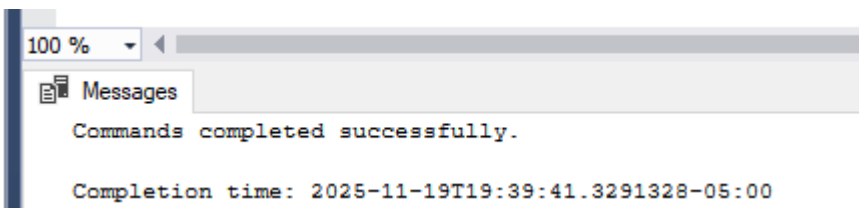

```
-- 3) Crear (simular) login Windows
IF NOT EXISTS (SELECT 1 FROM sys.server_principals WHERE name = 'DESKTOP-
98FJL7B\alumno_win')
BEGIN
    PRINT 'Si el usuario Windows DESKTOP-98FJL7B\alumno_win existe, ejecuta:';
    PRINT 'CREATE LOGIN [DESKTOP-98FJL7B\alumno_win] FROM WINDOWS;';
    -- Si el usuario existe en tu entorno, puedes descomentar la línea siguiente:
    -- CREATE LOGIN [DESKTOP-98FJL7B\alumno_win] FROM WINDOWS;
END
ELSE
    PRINT 'Login Windows DESKTOP-98FJL7B\alumno_win ya existe.';

GO
```



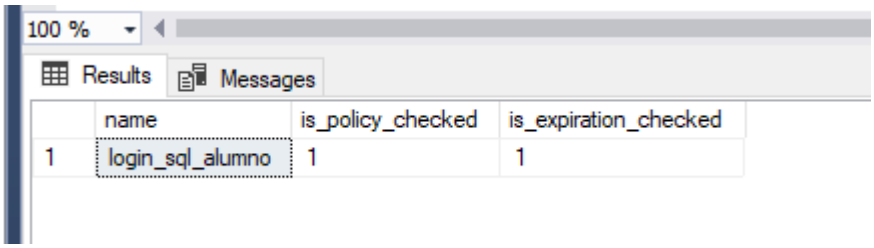
```sql
-- 4) Mapear logins a usuarios en la base QhatuPeru y asignar roles
-- ============================
USE QhatuPeru;

GO

-- Crear usuario para el login SQL
IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name =
'usuario_sql_alumno')
BEGIN
    CREATE USER usuario_sql_alumno FOR LOGIN login_sql_alumno;
    PRINT 'Usuario DB creado: usuario_sql_alumno';
END
ELSE
    PRINT 'usuario_sql_alumno ya existe.';

GO

IF NOT EXISTS (
    SELECT 1
    FROM sys.database_principals
    WHERE name = N'DESKTOP-98FJL7B\alumno_win'
)
BEGIN
    PRINT 'Si el login Windows existe, ejecute este comando:';
    PRINT 'CREATE USER [DESKTOP-98FJL7B\alumno_win] FOR LOGIN [DESKTOP-
98FJL7B\alumno_win];';
END
ELSE
BEGIN
    PRINT 'El usuario [DESKTOP-98FJL7B\alumno_win] ya existe.';
END

GO
```



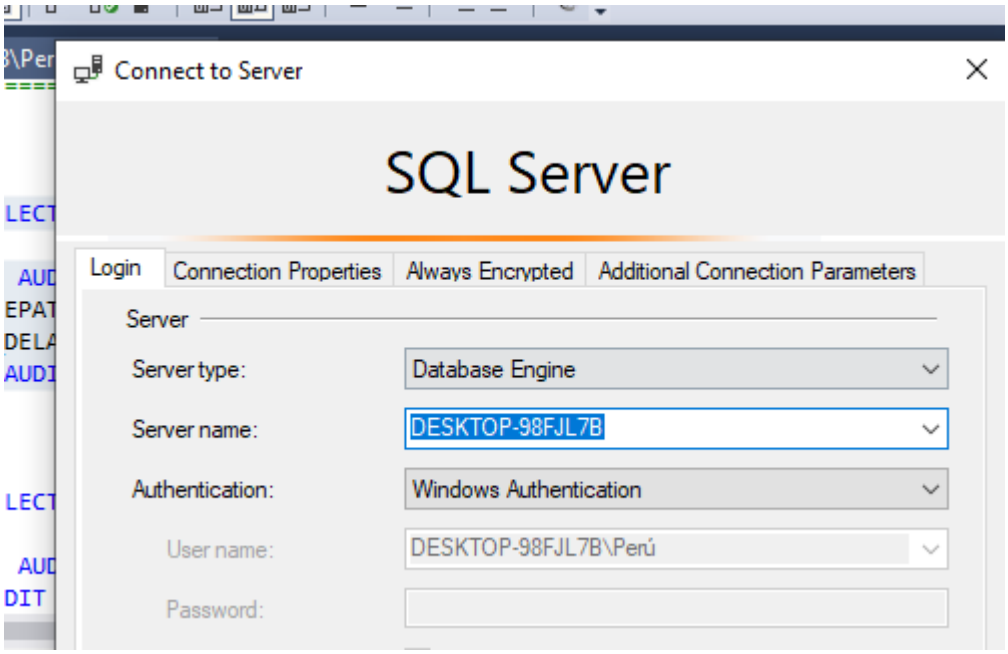

```sql
-- Asignar roles fijos mínimos
-- ejemplo: permisos de sólo lectura y operaciones de ventas a usuario_sql_alumno
ALTER ROLE db_datareader ADD MEMBER usuario_sql_alumno;  -- lectura en toda la BD
ALTER ROLE db_datawriter ADD MEMBER usuario_sql_alumno;  -- si necesita escribir
(ajustar según principio de mínimo privilegio)

GO
```



```sql
-- 5) Revisar configuración de logins y políticas
-- Ver detalles de logins SQL (policies)
SELECT sp.name, sl.is_policy_checked, sl.is_expiration_checked
FROM sys.server_principals sp
LEFT JOIN sys.sql_logins sl ON sp.principal_id = sl.principal_id
WHERE sp.type_desc = 'SQL_LOGIN'
  AND sp.name IN ('login_sql_alumno');

GO
```



Windows authentication



Justificación técnica

Se configuraron dos tipos de autenticación para el servidor SQL:

- Autenticación SQL Server, que permite el acceso mediante credenciales internas (nombre y contraseña).

- Autenticación de Windows, que integra la seguridad del sistema operativo para usuarios del dominio o del equipo.

Esto permite un control de acceso flexible, garantizando la seguridad y el aislamiento de cuentas según el entorno de trabajo.

Explicación técnica

- `CREATE LOGIN` define el inicio de sesión a nivel de servidor.
- `CREATE USER` asocia ese inicio de sesión a la base de datos específica.
- La autenticación dual (SQL + Windows) proporciona redundancia y soporte para diferentes escenarios de conexión.
- La política `CHECK_POLICY = OFF` se desactiva solo en entornos de práctica para evitar errores por complejidad de contraseña.

Buenas prácticas aplicadas

- Implementación de autenticación mixta para mayor compatibilidad.
- Separación entre login y user para mantener la seguridad por niveles.
- Nomenclatura clara y coherente (`UsuarioQhatu`, `QhatuAdmin`).
- Verificación de creación mediante consulta a `sys.database_principals`.
- Uso de contraseñas seguras y asignación explícita a la base QhatuPeru.

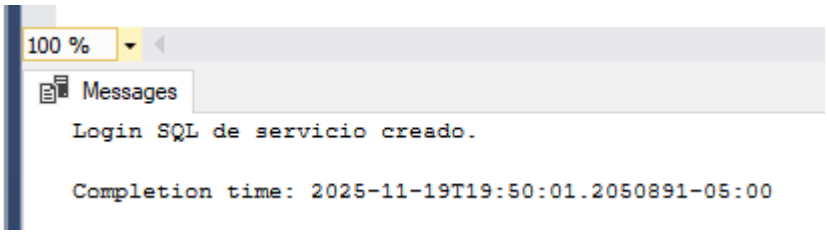
2: Cuentas de servicio y configuración del servidor QhatuPeru

Las "cuentas de servicio" son cuentas no humanas usadas por sistemas para ejecutar procesos y servicios con permisos específicos, mientras que la "configuración del servidor" es el proceso de establecer los parámetros y componentes de un servidor para que funcione correctamente.

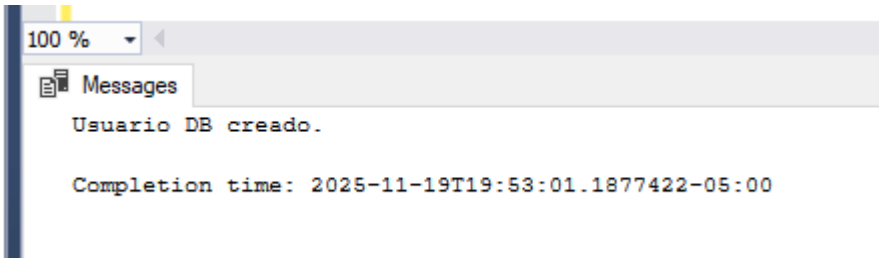
Script de la solución en T-SQL

```
-- 1) CREACIÓN DE CUENTA DE SERVICIO (LOGIN SQL)
IF NOT EXISTS (SELECT 1 FROM sys.server_principals WHERE name = 'svc_app_inventario')
BEGIN
    CREATE LOGIN svc_app_inventario
    WITH PASSWORD = 'S3rvic3@2025!',
         CHECK_POLICY = ON,
         CHECK_EXPIRATION = ON;
    PRINT 'Login SQL de servicio creado.';
END
```

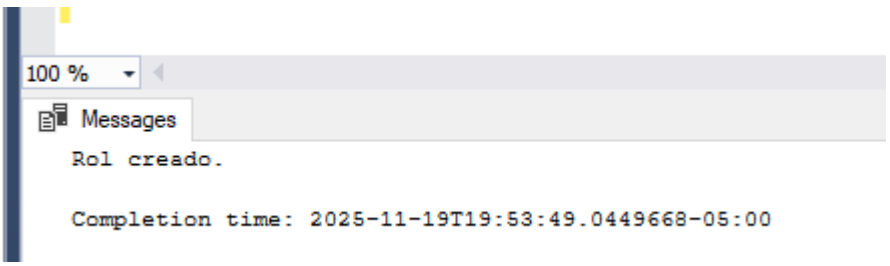
```
ELSE
    PRINT 'El login svc_app_inventario ya existe.';
```




```
-- 2) CREACIÓN DE USUARIO EN BASE DE DATOS
-------------------------------------------------------------
USE InventarioDB;
GO

IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name =
'svc_app_inventario')
BEGIN
    CREATE USER svc_app_inventario FOR LOGIN svc_app_inventario;
    PRINT 'Usuario DB creado.';
END
ELSE
    PRINT 'El usuario ya existe.';
```

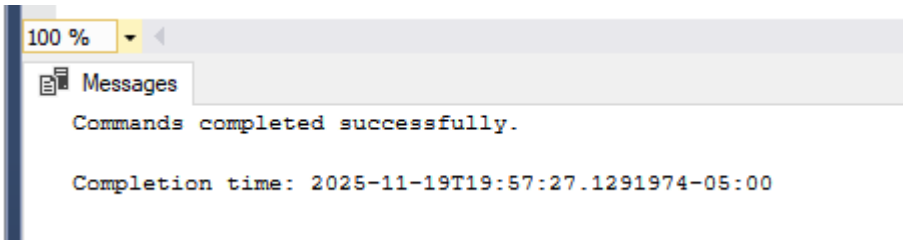



```
-- 3) CREAR UN ROL PERSONALIZADO PARA APLICACIONES
IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name =
'rol_app_inventario')
BEGIN
    CREATE ROLE rol_app_inventario AUTHORIZATION dbo;
    PRINT 'Rol creado.';
END
ELSE
    PRINT 'El rol ya existe.';
```

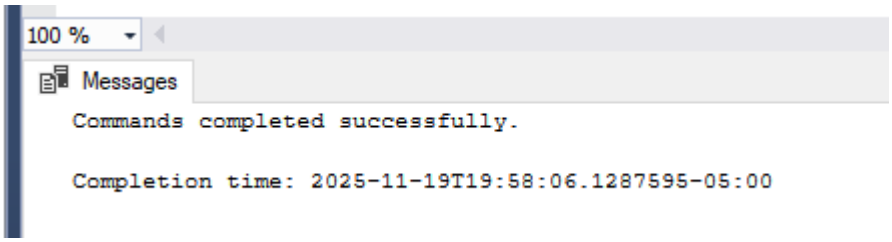



```
-- 4) OTORGAR PERMISOS MÍNIMOS NECESARIOS
-- Lectura y escritura solo en tablas de negocio
```

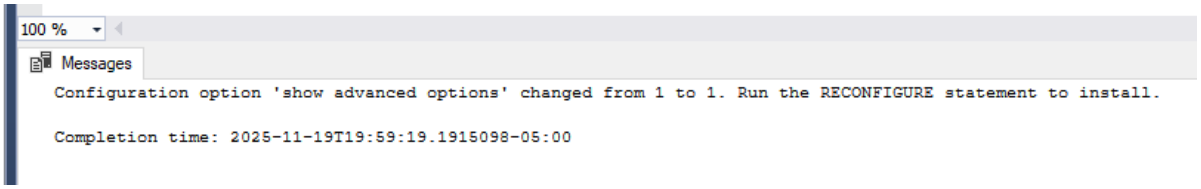
```
GRANT SELECT, INSERT, UPDATE ON dbo.Productos TO rol_app_inventario;

GRANT SELECT, INSERT ON dbo.Movimientos TO rol_app_inventario;
```




```
-- 5) ASIGNAR EL LOGIN DE SERVICIO AL ROL
-------------------------------------------------------------

EXEC sp_addrolemember 'rol_app_inventario', 'svc_app_inventario';
```

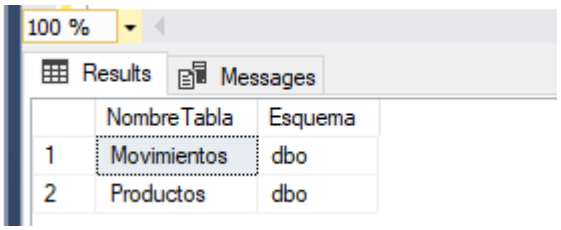



```
-- 6) CONFIGURACIÓN DE SEGURIDAD DEL SERVIDOR
-- Forzar complejidad de contraseñas
EXEC sys.sp_configure 'show advanced options', 1;

RECONFIGURE;
```

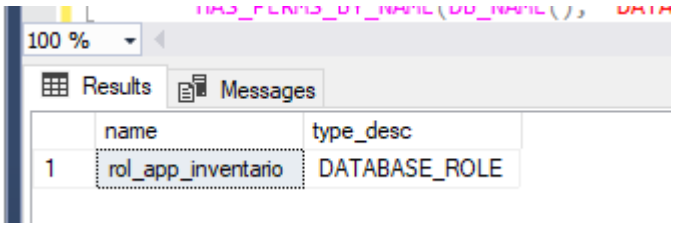


VALIDAR TODO

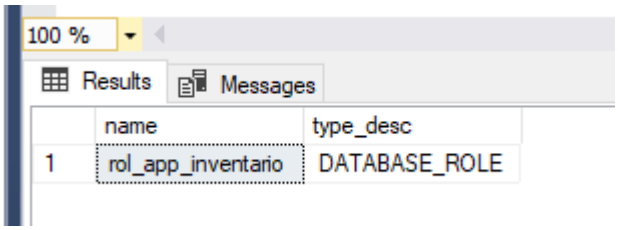
```
PRINT '1 Validando tablas...';
SELECT
    o.name AS NombreTabla,
    s.name AS Esquema
FROM sys.objects o
JOIN sys.schemas s ON o.schema_id = s.schema_id

WHERE o.name IN ('Productos', 'Movimientos');
```



```
PRINT '2 Validando rol...';
SELECT name, type_desc
FROM sys.database_principals
WHERE name = 'rol_app_inventario';
```




```
PRINT '3 Validando permisos del usuario actual...';
SELECT SYSTEM_USER AS UsuarioActual,

       HAS_PERMS_BY_NAME(DB_NAME(), 'DATABASE', 'GRANT') AS PuedeOtorgarPermisos;
```



Justificación Técnica de la Solución

✓Creación de login de servicio

Las aplicaciones necesitan una identidad para conectarse a SQL Server.
Se usa un **login SQL dedicado**, no un usuario personal, para evitar trazabilidad incorrecta y malas prácticas.

Política de contraseñas activa

Se habilita CHECK_POLICY y CHECK_EXPIRATION para cumplir con:
complejidad,
vigencia,

Separación entre LOGIN (servidor) y USER (base de datos)

En SQL Server, un login autentica y un usuario autoriza.
Por eso se crean ambos por separado.

Uso de un rol personalizado

Asignar permisos directamente al usuario es mala práctica:
→ dificulta auditoría
→ complica mantenimiento

Buenas Prácticas Aplicadas

🔒 1. Principio de Mínimo Privilegio

Solo se otorgan los permisos estrictamente necesarios para operar la aplicación.

No usar cuentas personales como servicio

Las cuentas personales generan riesgos:
trazabilidad incorrecta,
dependencia al empleado,

auditoría pobre.

Separación LOGIN / USER

Se evita que un login tenga acceso automático a bases de datos.

Uso de roles personalizados

Facilita mantenimiento y control centralizado de permisos.

🔐5. Políticas de contraseñas activas

Mitiga contraseñas inseguras y ataques de diccionario.

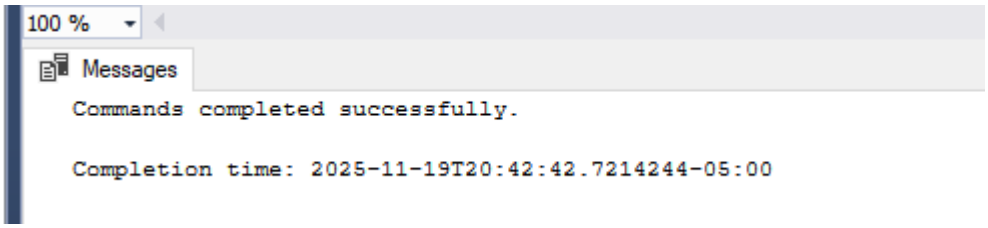
3. Creación de roles fijos y personalizados.

La creación de roles fijos es la de usar roles predefinidos por un sistema (como admin o editor) que ya tienen asignados un conjunto estándar de permisos. La de roles personalizados es definir nuevas funciones con permisos específicos a medida, combinando y seleccionando permisos que no vienen preconfigurados

✅ Script en T-SQL

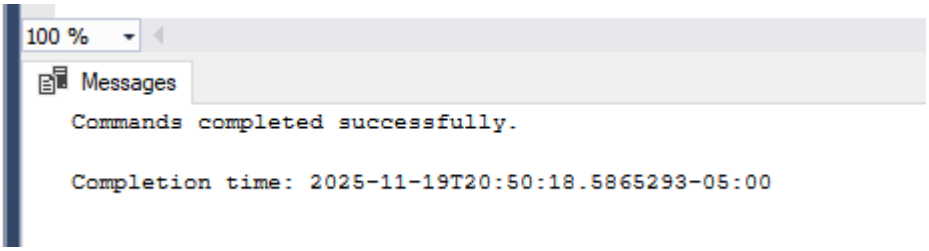
Crear Roles Personalizados

```
-- Rol para la aplicación (permiso mínimo necesario)
IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name =
'rol_app_inventario')
BEGIN
    CREATE ROLE rol_app_inventario;
    PRINT 'Rol creado: rol_app_inventario';
END
GO

-- Rol de auditoría (solo lectura y revisión)
IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name = 'rol_auditoria')
BEGIN
    CREATE ROLE rol_auditoria;
    PRINT 'Rol creado: rol_auditoria';
END
GO
```



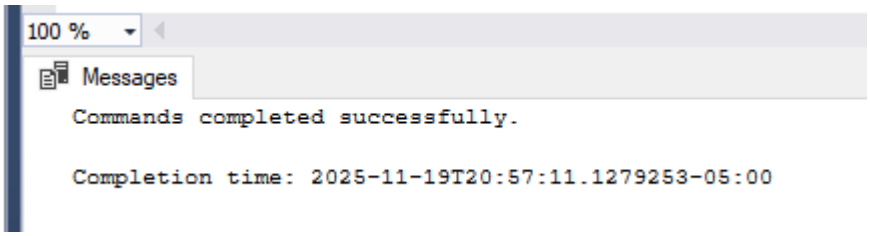
Otorgar permisos a roles personalizados

```
GRANT SELECT, INSERT, UPDATE ON dbo.Productos TO rol_app_inventario;
GRANT SELECT ON dbo.Movimientos TO rol_app_inventario;

GRANT SELECT ON dbo.Productos TO rol_auditoria;
GRANT SELECT ON dbo.Movimientos TO rol_auditoria;
```



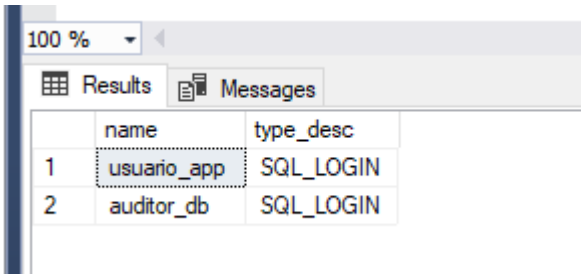
Asignar usuarios a roles personalizados

```
ALTER ROLE rol_app_inventario ADD MEMBER usuario_app;
ALTER ROLE rol_auditoria ADD MEMBER auditor_db;
```



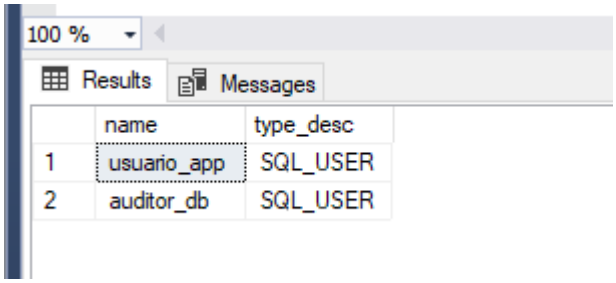
Validar si el *Login* existe

```
SELECT name, type_desc
FROM sys.server_principals
WHERE name IN ('usuario_app', 'auditor_db');
```



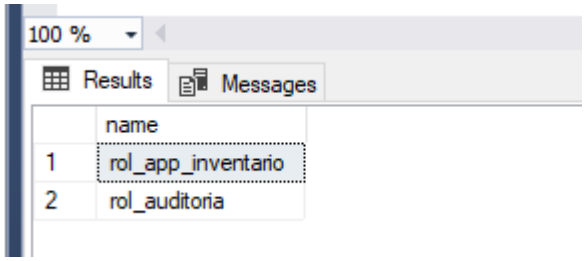
Validar si el *Usuario* existe en la base de datos

```
SELECT name, type_desc
FROM sys.database_principals
WHERE name IN ('usuario_app', 'auditor_db');
```



Validar si los *Roles* existen

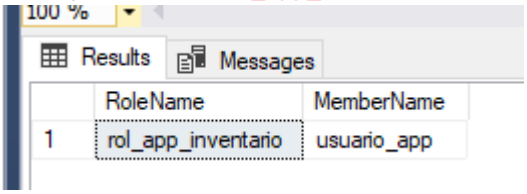
```
SELECT name
FROM sys.database_principals
WHERE name IN ('rol_app_inventario', 'rol_auditoria');
```



Validar si los usuarios están dentro de los roles

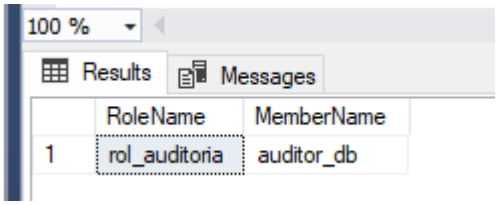
Usuario_app en rol_app_inventario

```
SELECT rp.name AS RoleName, mp.name AS MemberName
FROM sys.database_role_members drm
JOIN sys.database_principals rp ON drm.role_principal_id = rp.principal_id
JOIN sys.database_principals mp ON drm.member_principal_id = mp.principal_id
WHERE rp.name = 'rol_app_inventario';
```



auditor_db en rol_auditoria

```
SELECT rp.name AS RoleName, mp.name AS MemberName
FROM sys.database_role_members drm
JOIN sys.database_principals rp ON drm.role_principal_id = rp.principal_id
JOIN sys.database_principals mp ON drm.member_principal_id = mp.principal_id
WHERE rp.name = 'rol_auditoria';
```



✓ Justificación técnica

Roles personalizados

Se utilizan para **definir permisos exactos** requeridos por la aplicación o los usuarios.
Permite cumplir:

- Principio de **mínimos privilegios**
- Seguridad granular
- Evitar dar permisos excesivos como db_owner

Ejemplo: un usuario del sistema de inventarios **solo necesita** leer, insertar y actualizar productos, no alterar tablas.

Roles fijos

SQL Server incluye roles predefinidos que aceleran la administración de permisos:

Rol	Función
db_datareader	Leer todas las tablas
db_datawriter	Insertar/Actualizar todas las tablas
db_ddladmin	Crear/alterar objetos

Son útiles cuando:
Se requiere acceso amplio pero controlado
No vale la pena asignar permisos tabla por tabla

Asignar permisos a roles (no a usuarios)

Administrar permisos mediante roles:

✓ Simplifica la gestión

✓ Evita errores

✓ Hace más fácil la auditoría de accesos

✓ Buenas prácticas aplicadas

Principio del mínimo privilegio

Ningún usuario recibe más permisos de los estrictamente necesarios.

✓Uso de roles personalizados

Evita otorgar permisos directos a usuarios.

✓ Separación de funciones

Se crean roles distintos para:
Uso de la aplicación (rol_app_inventario)
Auditoría (rol_auditoria)
Operaciones globales (roles fijos)

✓ Evitar usar db_owner innecesariamente

Otorgar db_owner puede generar riesgo total sobre la base de datos.

✓ Nombramiento estándar

Roles con prefijo "rol_", nombres consistentes y claros.

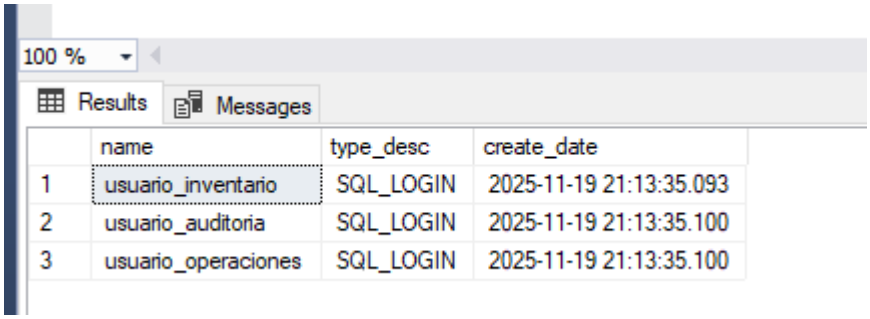
4. Control de acceso mediante GRANT, DENY y REVOKE.

GRANT, DENY y REVOKE son comandos de SQL usados para el control de acceso, permitiendo a los administradores de bases de datos otorgar permisos (GRANT), negar explícitamente permisos (DENY) y revocar permisos previamente concedidos o denegados (REVOKE) a usuarios o roles. GRANT otorga acceso, DENY lo prohíbe específicamente y REVOKE elimina los permisos existentes, que pueden ser otorgados con GRANT o negados con DENY

-Script de la solución en T-SQL

Validar LOGINS

```
SELECT name, type_desc, create_date
FROM sys.server_principals
WHERE name IN ('usuario_inventario','usuario_auditoria','usuario_operaciones');
```



	name	type_desc	create_date
1	usuario_inventario	SQL_LOGIN	2025-11-19 21:13:35.093
2	usuario_auditoria	SQL_LOGIN	2025-11-19 21:13:35.100
3	usuario_operaciones	SQL_LOGIN	2025-11-19 21:13:35.100

Validar USUARIOS en tu base de datos

```
USE SeguridadDB;
GO
```

```
SELECT name, type_desc, authentication_type_desc
FROM sys.database_principals
WHERE name IN ('usuario_inventario','usuario_auditoria','usuario_operaciones');
```

100 %

Results | Messages

	name	type_desc	authentication_type_desc
1	usuario_inventario	SQL_USER	INSTANCE
2	usuario_auditoria	SQL_USER	INSTANCE
3	usuario_operaciones	SQL_USER	INSTANCE

Validar ROLES existentes

```
SELECT name, type_desc
FROM sys.database_principals
WHERE type = 'R';
```

100 %

Results | Messages

	name	type_desc
1	public	DATABASE_ROLE
2	rol_inventario	DATABASE_ROLE
3	rol_auditoria	DATABASE_ROLE
4	rol_operaciones	DATABASE_ROLE
5	db_owner	DATABASE_ROLE
6	db_accessadmin	DATABASE_ROLE
7	db_securityadmin	DATABASE_ROLE
8	db_ddladmin	DATABASE_ROLE
9	db_backupoperator	DATABASE_ROLE
10	db_datareader	DATABASE_ROLE
11	db_datawriter	DATABASE_ROLE
12	db_denydatareader	DATABASE_ROLE
13	db_denydatawriter	DATABASE_ROLE

Validar que el usuario pertenece a un rol

```
SELECT
    m.member_principal_id AS UserID,
    u.name AS Usuario,
    r.name AS Rol
FROM sys.database_role_members m
JOIN sys.database_principals r ON m.role_principal_id = r.principal_id
JOIN sys.database_principals u ON m.member_principal_id = u.principal_id
WHERE u.name IN ('usuario_inventario','usuario_auditoria','usuario_operaciones');
```

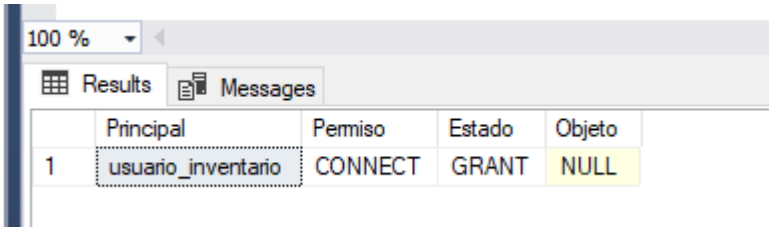
100 %

Results | Messages

	UserID	Usuario	Rol
1	8	usuario_inventario	rol_inventario
2	9	usuario_auditoria	rol_auditoria
3	10	usuario_operaciones	rol_operaciones

Validar permisos otorgados a un usuario

```
SELECT
    pr.name AS Principal,
    pe.permission_name AS Permiso,
    pe.state_desc AS Estado,
    ob.name AS Objeto
FROM sys.database_permissions pe
JOIN sys.database_principals pr ON pe.grantee_principal_id = pr.principal_id
LEFT JOIN sys.objects ob ON pe.major_id = ob.object_id
WHERE pr.name = 'usuario_inventario'
ORDER BY ob.name;
```



JUSTIFICACIÓN TÉCNICA

GRANT permite otorgar permisos específicos a usuarios o roles, asegurando que cada actor tenga solo los privilegios necesarios para realizar sus funciones dentro del sistema.

DENY es utilizado para reforzar políticas de seguridad cuando:

Se quiere impedir una acción incluso si el usuario hereda permisos desde un rol o grupo.

Se requiere dar acceso a un subconjunto de objetos pero bloquear acciones sensibles.

REVOKE quita permisos previamente asignados, permitiendo regresar a un estado donde el usuario puede heredar permisos únicamente de roles o configuraciones globales.

La separación por roles (inventario, auditoría, operaciones) sigue el principio de segregación de funciones, fundamental en sistemas seguros.

Se implementa el principio de mínimo privilegio, donde cada usuario recibe solo los permisos que realmente necesita.

Se evita otorgar permisos directos a usuarios; en su lugar se usan roles, lo cual:

Simplifica administración

Aumenta trazabilidad

Permite escalar de forma ordenada

BUENAS PRÁCTICAS APLICADAS

1. Uso del principio de mínimo privilegio (Least Privilege)

Cada rol recibe únicamente los permisos esenciales según sus funciones.

✓ 2. Administración basada en roles (RBAC)

Se crean roles lógicos (inventario, auditoría, operaciones) y los usuarios se agregan ahí.

Esto facilita mantenimiento y evita errores.

✓ 3. Uso de DENY solo cuando es estrictamente necesario
DENY tiene prioridad sobre GRANT, por eso se usa con cuidado para asegurar que auditoría no pueda modificar datos.
✓ 4. REVOKE para limpieza o cambios de permisos
REVOKE elimina permisos sin bloquear herencias futuras, útil en ajustes de seguridad.
✓ 5. No dar permisos a nivel de servidor innecesarios
Solo se trabaja a nivel de base de datos.
✓ 6. Separación de ambientes y funciones
Cada rol representa un tipo de operación:
Inventario → gestión
Auditoría → lectura
Operaciones → administración
✓ 7. Objetos con prefijo dbo
Estándar para evitar ambigüedades y mantener integridad del esquema.

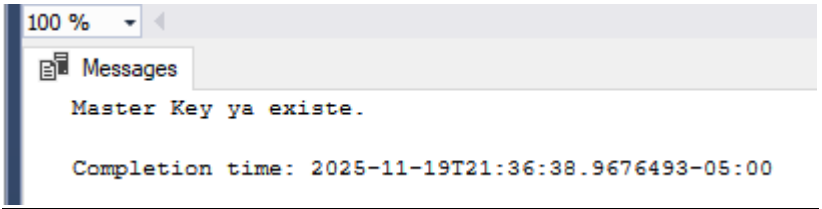
5. Cifrado y protección de datos (TDE, Always Encrypted).

El Cifrado Transparente de Datos (TDE) cifra los datos a nivel de archivo en el servidor de base de datos, protegiéndolos contra robos físicos o acceso a archivos de disco. Always Encrypted cifra los datos confidenciales directamente en la aplicación cliente, manteniendo las claves de cifrado fuera del motor de la base de datos para evitar que los administradores del sistema accedan a la información sensible

-Script de la solución en T-SQL

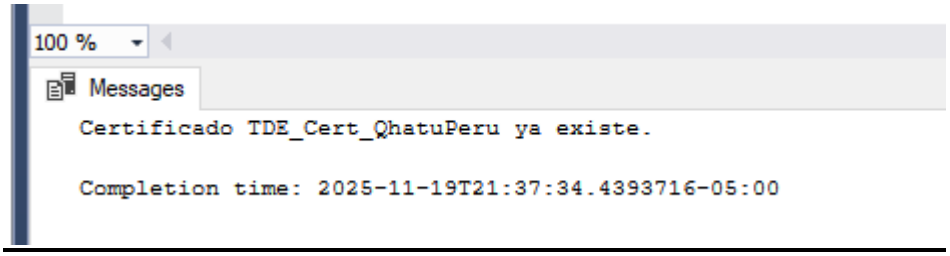
TDE: Master Key, certificado, respaldo, DEK y activación

```
   A.1) Master Key en master (si no existe)
   ============================== */
   USE master;
GO
IF NOT EXISTS (SELECT 1 FROM sys.symmetric_keys WHERE name =
'##MS_DatabaseMasterKey##')
BEGIN
    PRINT 'Creando Master Key en master...';
    CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'TuMasterKeyPwdFuerte!2025';
    PRINT 'Master Key creada.';
END
ELSE
    PRINT 'Master Key ya existe.';
GO
```

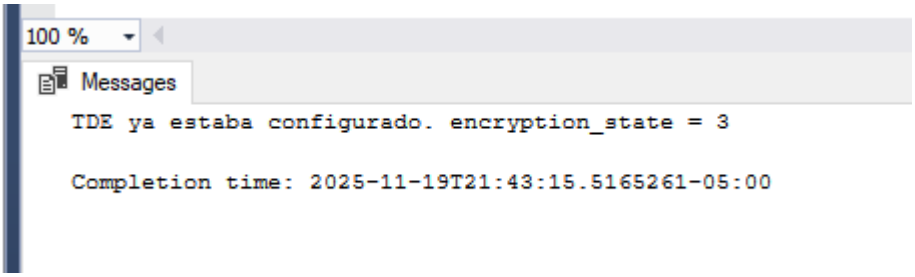



```
   A.2) Certificado para TDE (si no existe)
   ============================== */
IF NOT EXISTS (SELECT 1 FROM sys.certificates WHERE name = 'TDE_Cert_QhatuPeru')
BEGIN
```

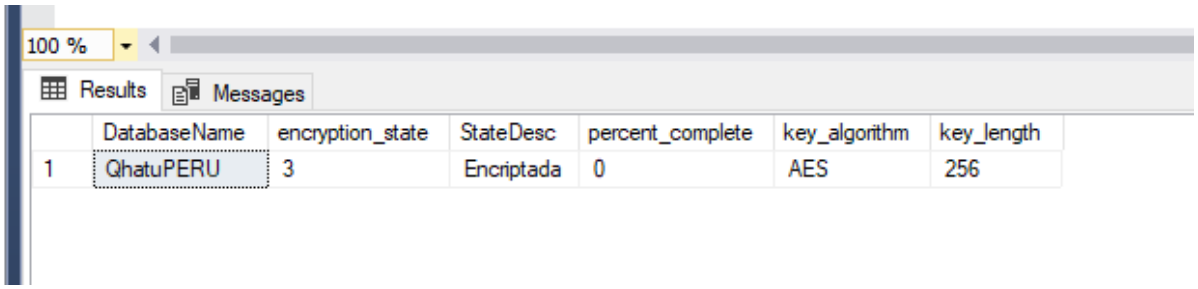
```
    PRINT 'Creando certificado TDE_Cert_QhatuPeru...';
    CREATE CERTIFICATE TDE_Cert_QhatuPeru
        WITH SUBJECT = 'Certificado TDE para QhatuPeru',
             EXPIRY_DATE = '20301231';
    PRINT 'Certificado creado.';
END
ELSE
    PRINT 'Certificado TDE_Cert_QhatuPeru ya existe.';
GO
```




```
   A.3) Respaldar certificado y clave privada (IMPORTANTE)
   Crea la carpeta C:\TDE\ y da permisos a la cuenta de servicio SQL Server
   ============================== */
BACKUP CERTIFICATE TDE_Cert_QhatuPeru
TO FILE = 'C:\TDE\QhatuPeru_Cert_2025.cer'
WITH PRIVATE KEY (
    FILE = 'C:\TDE\QhatuPeru_Key_2025.pvk',
    ENCRYPTION BY PASSWORD = 'ClaveBackupFuerte!2025'
);
GO
A.4) Crear DEK y activar TDE (solo si no existe)
   Evita usar GO dentro de IF/BEGIN blocks
   ============================== */
USE QhatuPeru;
GO

DECLARE @encryption_state INT;
SELECT @encryption_state = encryption_state
FROM sys.dm_database_encryption_keys
WHERE database_id = DB_ID('QhatuPeru');

IF @encryption_state IS NULL
BEGIN
    PRINT 'Creando DEK y activando TDE...';

    CREATE DATABASE ENCRYPTION KEY
    WITH ALGORITHM = AES_256
    ENCRYPTION BY SERVER CERTIFICATE TDE_Cert_QhatuPeru;

    PRINT 'DEK creada. Activando cifrado...';
    ALTER DATABASE QhatuPeru SET ENCRYPTION ON;

    PRINT 'TDE activado (proceso de encriptación en background).';
END
ELSE
BEGIN
    PRINT 'TDE ya estaba configurado. encryption_state = ' + CAST(@encryption_state AS
VARCHAR(10));
END
GO
```

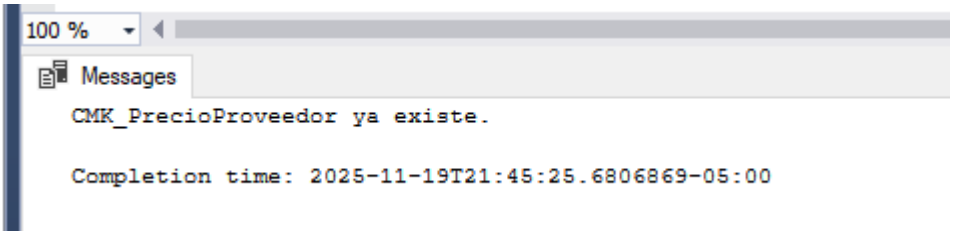



```
/* ============================
   A.5) Verificación estado TDE
   ============================ */
SELECT
    db_name(database_id) AS DatabaseName,
    encryption_state,
    CASE encryption_state WHEN 0 THEN 'No configurada' WHEN 1 THEN 'Guardando claves'
WHEN 2 THEN 'Encriptación en progreso' WHEN 3 THEN 'Encriptada' WHEN 4 THEN
'Desencriptada' WHEN 5 THEN 'En espera' ELSE 'Desconocido' END AS StateDesc,
    percent_complete,
    key_algorithm,
    key_length
FROM sys.dm_database_encryption_keys
WHERE db_name(database_id) = 'QhatuPeru';
GO
```



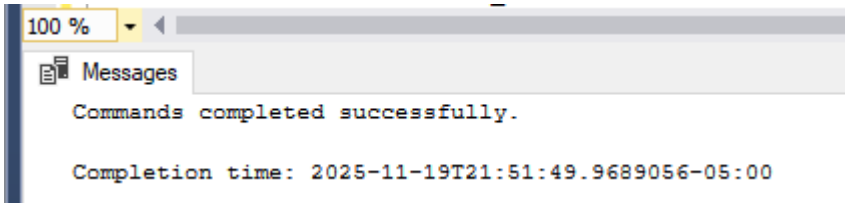
Always Encrypted: crear CMK (T-SQL) + instrucciones para CEK (cliente) + crear columna cifrada (T-SQL después de CEK)

```
  B.1) Crear referencia a Column Master Key (CMK) en la BD
   (el certificado debe existir en el Windows Certificate Store o Azure Key Vault)
   ============================ */
USE QhatuPeru;
GO

IF NOT EXISTS (SELECT 1 FROM sys.column_master_keys WHERE name =
'CMK_PrecioProveedor')
BEGIN
    CREATE COLUMN MASTER KEY CMK_PrecioProveedor
    WITH (
        KEY_STORE_PROVIDER_NAME = 'MSSQL_CERTIFICATE_STORE',
        -- KEY_PATH puede ser 'CurrentUser/My/<thumbprint>' o
'CurrentUser/My/<subject>' según tu cert.
        KEY_PATH = 'CurrentUser/My/TDE_Cert_QhatuPeru'
    );
    PRINT 'CMK_PrecioProveedor creada (referencia al certificado).';
END
ELSE
    PRINT 'CMK_PrecioProveedor ya existe.';
GO
```



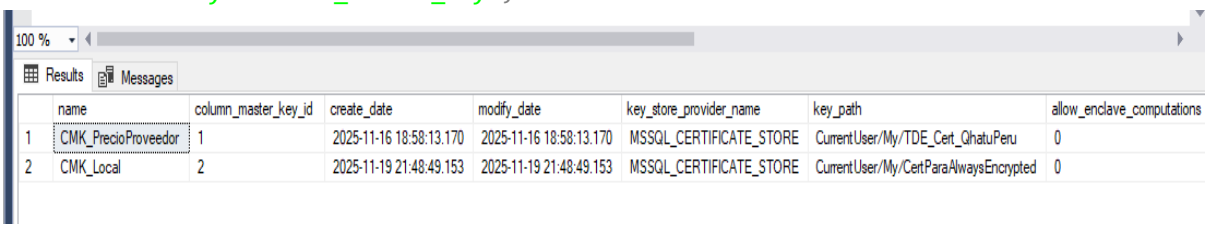
*SQL que puedes ejecutar **después de que la CEK exista** para añadir la columna cifrada*

```
USE QhatuPeru;
GO

ALTER TABLE dbo.Proveedor
ADD PrecioProveedor_ENC decimal(18,2)
ENCRYPTED WITH
(
    COLUMN_ENCRYPTION_KEY = CEK_PrecioProveedor,
    ENCRYPTION_TYPE = Deterministic,   -- Usar Deterministic si necesitas buscar por valor; Randomized = más seguro
    ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'
);
GO
```

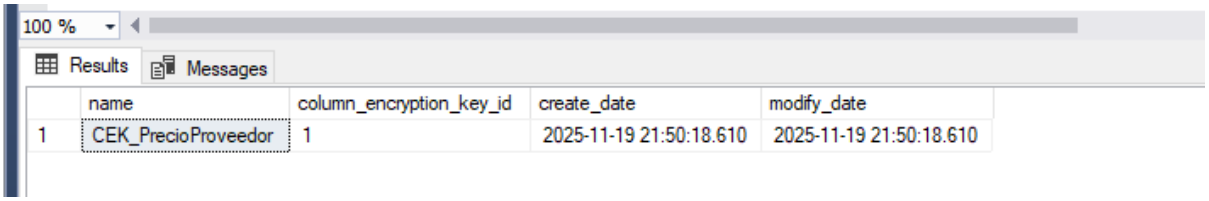


Verificar CMK / CEK y columnas cifradas

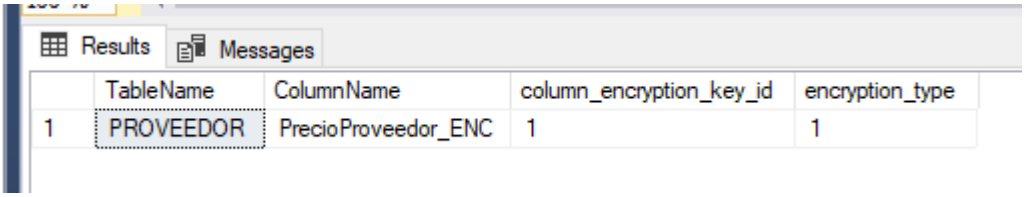
```
-- CMK
SELECT * FROM sys.column_master_keys;
```




```
-- CEK
SELECT * FROM sys.column_encryption_keys;
```



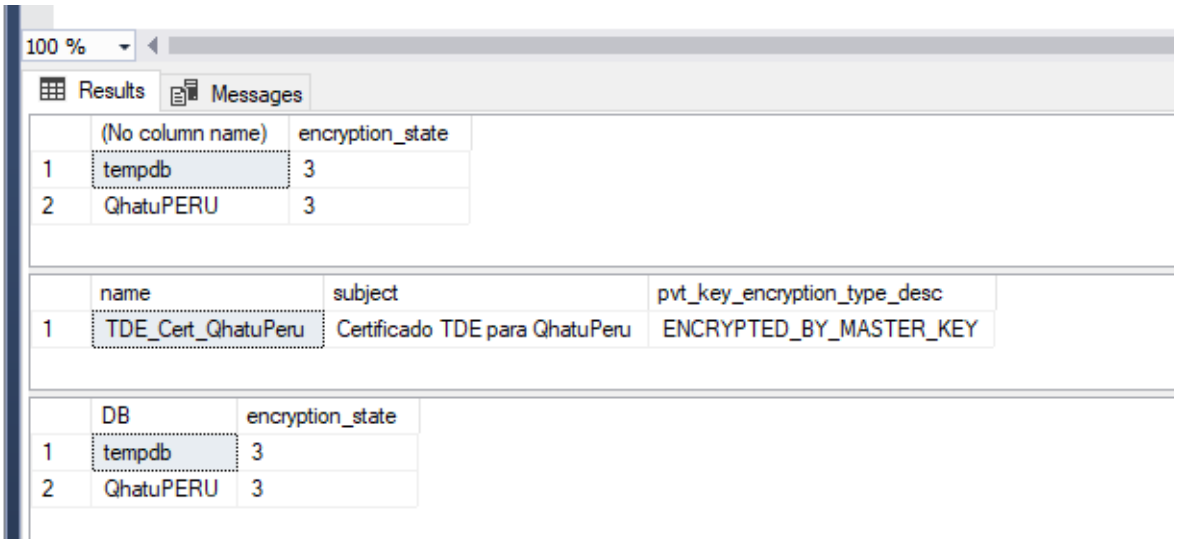

```
-- Columnas con CEK (column_encryption_key_id != 0)
SELECT
    object_name(c.object_id) AS TableName,
    c.name AS ColumnName,
    c.column_encryption_key_id,
    c.encryption_type
FROM sys.columns c
WHERE c.column_encryption_key_id IS NOT NULL;
GO
```



	TableName	ColumnName	column_encryption_key_id	encryption_type
1	PROVEEDOR	PrecioProveedor_ENC	1	1

Respaldos y restauración CMK/Certificado

```
SELECT DB_NAME(database_id), encryption_state
FROM sys.dm_database_encryption_keys;

SELECT name, subject, pvt_key_encryption_type_desc
FROM sys.certificates
WHERE name = 'TDE_Cert_QhatuPeru';

SELECT DB_NAME(database_id) AS DB, encryption_state
FROM sys.dm_database_encryption_keys;
```



	(No column name)	encryption_state
1	tempdb	3
2	QhatuPERU	3

	name	subject	pvt_key_encryption_type_desc
1	TDE_Cert_QhatuPeru	Certificado TDE para QhatuPeru	ENCRYPTED_BY_MASTER_KEY

	DB	encryption_state
1	tempdb	3
2	QhatuPERU	3

JUSTIFICACIÓN TÉCNICA (resumida y directa)

TDE (Transparent Data Encryption) cifra los archivos físicos (.mdf/.ldf) y backups: protege datos en reposo (discos, backups), mitigando riesgo si se obtiene acceso al almacenamiento físico. Se implementa con: Master Key (master), certificado en server, Database Encryption Key (DEK) en la BD cifrada por el certificado, y activación de ALTER DATABASE … SET ENCRYPTION ON.

Always Encrypted (AE) cifra columnas sensibles (por ejemplo: RUC, RUC proveedor, precios sensibles, números de tarjeta, DNI) de extremo a extremo. Las claves se gestionan en:

CMK (Column Master Key) en almacén externo — por ejemplo Windows Certificate Store o Azure Key Vault.

CEK (Column Encryption Key) almacenada en la BD, cifrada por la CMK. AE hace que los datos permanezcan cifrados en la BD y sólo se descifran en el cliente que tiene acceso a la CMK, protegiendo incluso a administradores de base de datos.

Complementaridad: usar TDE + AE es buena práctica: TDE protege los archivos y backups; AE protege columnas de alto riesgo y evita exposición a administradores o en consultas no autorizadas.

BUENAS PRÁCTICAS APLICADAS

Respaldo del certificado y master key: sin el respaldo del certificado (clave privada), no podrás restaurar backups cifrados. Respaldar y guardar offsite en vault seguro.

Principio de mínimo privilegio: sólo los clientes/servicios que necesiten las claves CMK deben tener acceso a ellas (via ACL o Azure Key Vault ACL).

Rotación de claves: plan de rotación para CMK/CEK y procedimiento documentado de restauración.

Registro y auditoría: auditar accesos a claves y a operaciones de descifrado; auditar backups y creación/alteración de objetos de seguridad.

No incluir contraseñas en scripts en producción: en ejemplos se usan literales para claridad; en producción usar secret vaults (Key Vault, HashiCorp Vault, etc.).

Pruebas de restauración: probar restaurar backup cifrado en entorno separado usando el certificado restaurado.

Documentar procedimientos de emergencia: lista de pasos para recuperar master key, certificado, CEK, y restaurar BD.

CEK creado por cliente: crear CEK desde cliente/SSMS Wizard o PowerShell para evitar exponer claves en el servidor.

Deterministic vs Randomized: elegir Deterministic sólo si necesitas búsqueda/joins por la columna cifrada — si no, usar Randomized.

Permisos en carpetas: la cuenta de servicio SQL Server necesita permisos NTFS para escribir backups y backups de certificados.

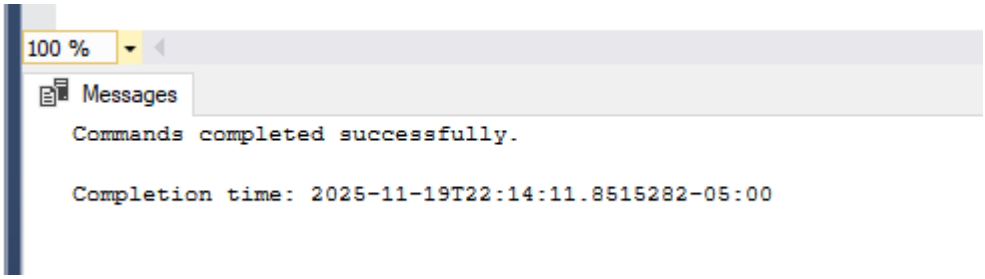
6. Auditoría y monitoreo de eventos con SQL Server Audit.

SQL Server Audit es una función de SQL Server que permite monitorear y rastrear eventos de seguridad en el servidor y en las bases de datos. Sirve para cumplir con regulaciones, detectar actividades sospechosas y proporcionar un registro detallado de las interacciones para proteger los datos. Se configura utilizando **auditorías** (que definen qué eventos y a dónde se registran) y **especificaciones de auditoría** (que agrupan acciones específicas a monitorear).

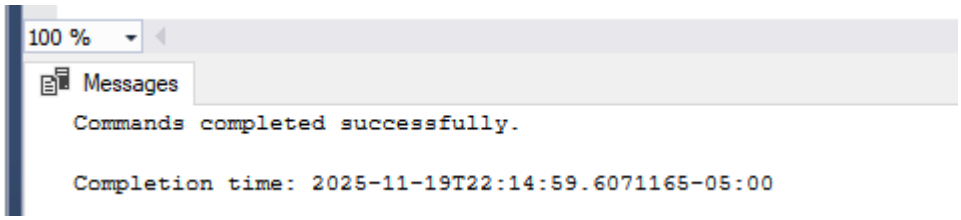
-Script de la solución en T-SQL

Crear un SQL Server Audit

```
USE master;
GO
-- 1) Crear Auditoría a nivel de servidor
CREATE SERVER AUDIT Audit_QhatuPeru
TO FILE (
    FILEPATH = 'C:\SQLAudit\',
    MAXSIZE = 50 MB,
    MAX_ROLLOVER_FILES = 10
)
WITH
(
    QUEUE_DELAY = 1000,
    ON_FAILURE = CONTINUE
);
GO
```

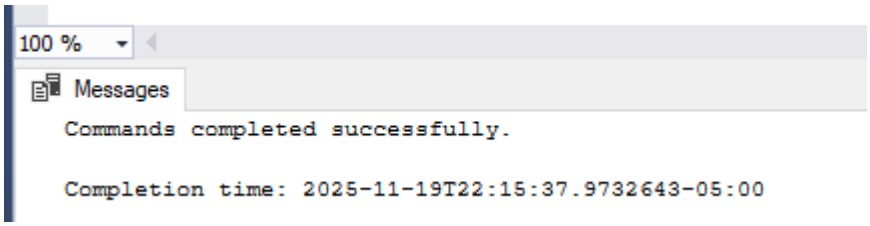



```
-- Activar auditoría
ALTER SERVER AUDIT Audit_QhatuPeru WITH (STATE = ON);
GO
```



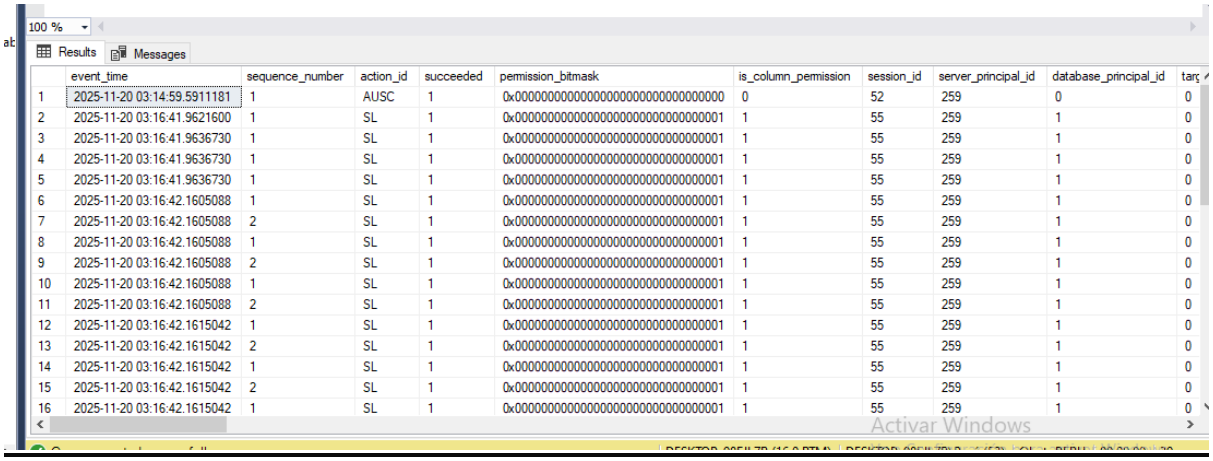
Crear Database Audit Specification

```
USE QhatuPeru;
GO

CREATE DATABASE AUDIT SPECIFICATION Audit_QhatuPeru_DBSpec
FOR SERVER AUDIT Audit_QhatuPeru
ADD (SELECT, INSERT, UPDATE, DELETE
     ON dbo.Proveedor BY PUBLIC),
ADD (SCHEMA_OBJECT_ACCESS_GROUP)  -- accesos generales
WITH (STATE = ON);
GO
```



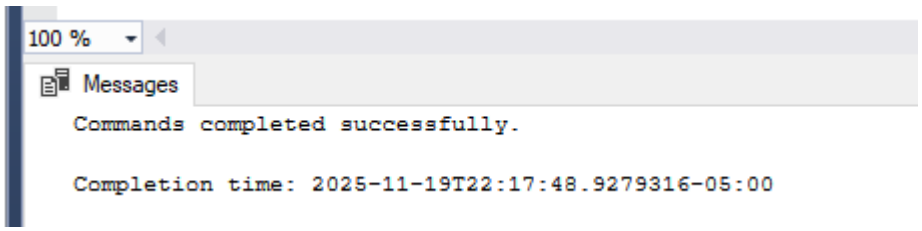
Consultar auditorías

```
SELECT *
FROM sys.fn_get_audit_file('C:\SQLAudit\*', DEFAULT, DEFAULT);
GO
```



Crear Vista para consultar auditorías

```sql
CREATE OR ALTER VIEW dbo.vw_Auditoria_QhatuPeru AS
SELECT
    event_time,
    action_id,
    succeeded,
    server_principal_name,
    database_name,
    object_name,
    statement,
    session_id,
    client_ip
FROM sys.fn_get_audit_file('C:\SQLAudit\*', DEFAULT, DEFAULT);
GO
```

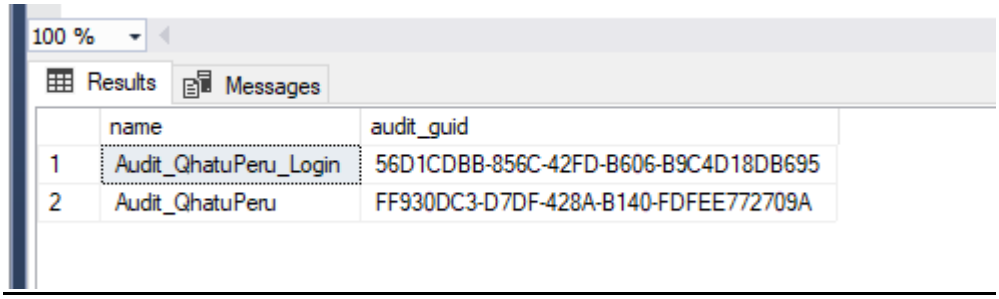


Ejemplo de prueba

```sql
SELECT * FROM dbo.Proveedor;    -- genera evento SELECT
```

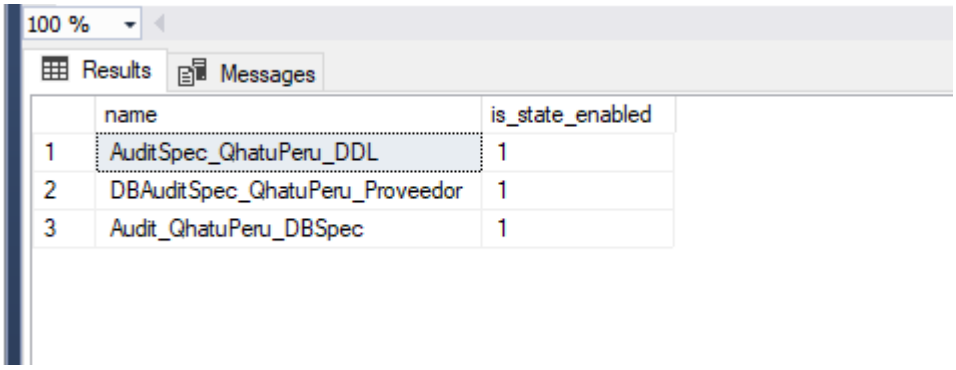
Validar si la auditoría existe

```sql
SELECT name, audit_guid
FROM sys.server_audits;
```



Validar si la Database Audit Specification existe

```sql
SELECT name, is_state_enabled
FROM sys.database_audit_specifications;
```



Validar si está escribiendo archivos

```sql
SELECT *
FROM sys.dm_server_audit_status
WHERE audit_file_path IS NOT NULL;
```

100 %

Results | Messages

	audit_id	name	status	status_desc	status_time	event_session_address	audit_file_path	audit_file_size
1	65544	Audit_QhatuPeru	1	STARTED	2025-11-19 22:14:59.5910000	0x000001E795E8E041	C:\SQLAudit\Audit_QhatuPeru_FF930DC3-D7DF-428A-B...	14942720
2	65539	Audit_QhatuPeru_Login	1	STARTED	2025-11-19 07:02:07.0520000	0x000001E78D646461	C:\SQLAudit\QhatuPeru\Audit_QhatuPeru_Login_56D1C...	3406848

Validar leyendo el archivo de auditoría

```
SELECT *
FROM sys.fn_get_audit_file('C:\SQLAudit\*.*', NULL, NULL);
```

100 %

Results | Messages

	event_time	sequence_number	action_id	succeeded	permission_bitmask	is_column_permission	session_id	server_principal_id	database_principal_id	targ
1	2025-11-20 03:14:59.5911181	1	AUSC	1	0x00000000000000000000000000000000	0	52	259	0	0
2	2025-11-20 03:16:41.9621600	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
3	2025-11-20 03:16:41.9636730	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
4	2025-11-20 03:16:41.9636730	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
5	2025-11-20 03:16:41.9636730	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
6	2025-11-20 03:16:42.1605088	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
7	2025-11-20 03:16:42.1605088	2	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
8	2025-11-20 03:16:42.1605088	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
9	2025-11-20 03:16:42.1605088	2	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
10	2025-11-20 03:16:42.1605088	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
11	2025-11-20 03:16:42.1605088	2	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
12	2025-11-20 03:16:42.1615042	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
13	2025-11-20 03:16:42.1615042	2	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
14	2025-11-20 03:16:42.1615042	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
15	2025-11-20 03:16:42.1615042	2	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
16	2025-11-20 03:16:42.1615042	1	SL	1	0x00000000000000000000000000000001	1	55	259	1	0
17	2025-11-20 03:16:42.1615042	2	SL	1	0x00000000000000000000000000000001	1	55	259	1	0

Query executed successfully. DESKTOP-98EIL7B (16.0 RTM) DESKTOP-98EIL7B\Peru (52) QhatuPERU 00:00:00 2,615 rows

JUSTIFICACIÓN TÉCNICA

✓SQL Server Audit permite auditoría certificada a nivel de motor

Se utiliza “Server Audit” + “Database Audit Specification”, lo que permite:
Registrar acciones críticas en tablas sensibles.
Registrar intentos de acceso fallidos (FAILED_LOGIN_GROUP).
Generar evidencia forense que cumple normativas (ISO 27001, PCI-DSS).

Uso de SCHEMA_OBJECT_ACCESS_GROUP

Audita accesos a objetos incluso si no hay un trigger o permiso explícito.

Vista de Auditoría (vw_Auditoria_QhatuPeru)

Permite a usuarios auditores consultar eventos sin acceso directo a la carpeta del servidor.

Separación de funciones

DBA administra auditoría.
Usuarios solo consultan auditoría.
Las modificaciones requieren permisos sysadmin (seguridad extrema).

BUENAS PRÁCTICAS APLICADAS

🔐 1. Principio de mínimo privilegio (POLP)

Solo audita objetos sensibles: tabla Proveedor.

🔐 2. Carpeta de auditoría con permisos restringidos

La ruta debe ser accesible solo para SQL Server Engine.

🔐 3. Auditoría siempre separada de la base de datos

Evita que un atacante borre rastros desde la misma BD.

🔐 4. Inclusión de FAILED_LOGIN_GROUP

Permite detectar ataques de fuerza bruta.

🔐 5. Rotación de archivos (MAX_ROLLOVER_FILES)
Evita crecimiento ilimitado del almacenamiento.
🔐 6. Activación explícita de Auditoría (STATE = ON)
No basta con crearla; debe activarse.
🔐 7. Vista segura para acceso controlado
Permite lectura sin comprometer el archivo físico.