Universidad Peruana Los Andes

Facultad de Ingeniería

Escuela profesional de Ingeniería de Sistemas

PRACTICA SEMANA 11

Curso: Base de datos II

Docente: Raul Enrique Fernandez Bejarano

Estudiante: Sarmiento Mosquera Yeims Abraham

Ciclo: V - Código: s03807f

Huancayo - 2025

EJERCICIOS PRACTICOS DE LA SEMANA 11

Enunciado 1

Proyecto 1: Autenticación: Comparación segura y configuración de logins

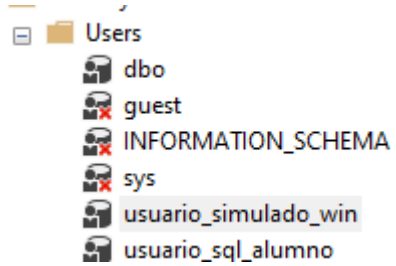
1. Enunciado del ejercicio

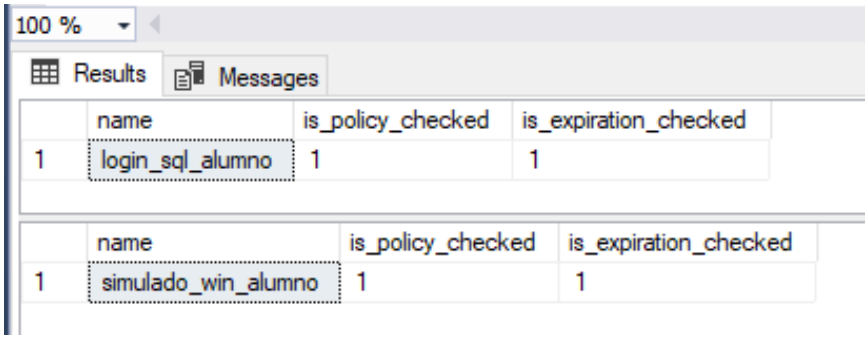
   Crear en el servidor dos logins de prueba: uno con autenticación SQL (login_sql_alumno) y otro que represente un usuario Windows (DOMAIN\alumno_win — simulado), aplicar políticas de contraseñas y mapear ambos a usuarios en la base QhatuPeru. Mostrar cómo forzar expiración y comprobar la política de contraseñas.

Script de la solución

```
-- Crear login con autenticación SQL
CREATE LOGIN login_sql_alumno
WITH PASSWORD = 'ContraseñaSegura123!',
     CHECK_POLICY = ON,        -- Aplica política de contraseñas
     CHECK_EXPIRATION = ON;    -- Fuerza expiración

-- Crear login que simula un usuario Windows
CREATE LOGIN simulado_win_alumno
WITH PASSWORD = 'SimulacionSegura2024!',
     CHECK_POLICY = ON,
     CHECK_EXPIRATION = ON;

-- Mapearlo a la base QhatuPeru
USE QhatuPeru

-- Crear usuarios mapeados a los logins
CREATE USER usuario_sql_alumno FOR LOGIN login_sql_alumno;
CREATE USER usuario_win_alumno FOR LOGIN simulado_win_alumno;

-- Verificar políticas aplicadas
SELECT name, is_policy_checked, is_expiration_checked
FROM sys.sql_logins
WHERE name = 'login_sql_alumno';

SELECT name, is_policy_checked, is_expiration_checked
FROM sys.sql_logins
WHERE name = 'simulado_win_alumno';
```

Resultado





Justificación de la técnica aplicada

Usamos CREATE LOGIN para registrar credenciales a nivel del servidor, además usamos CHECK_POLICY=ON que activa las políticas de contraseñas de windows

Con el comando CHECK_EXPIRATION=ON hace que los usuarios tengan que cambiar sus contraseñas cada cierto tiempo.

Por último, el comando sys.sql_logins nos permite ver si las políticas están correctamente activadas

Explicación de las buenas prácticas utilizadas

- El uso de las políticas de caracteres de contraseñas y de cambio de contraseñas cada cierto tiempo brinda mejor seguridad y evitan accesos prolongados sin renovación
- Se sigue el principio de menor privilegio con la separación entre el login y el usuario
- La validación con el ultimo comando nos permite asegurarnos que se están aplicando las políticas

Enunciado 2

Proyecto 2: Cuentas de servicio y configuración segura del servidor

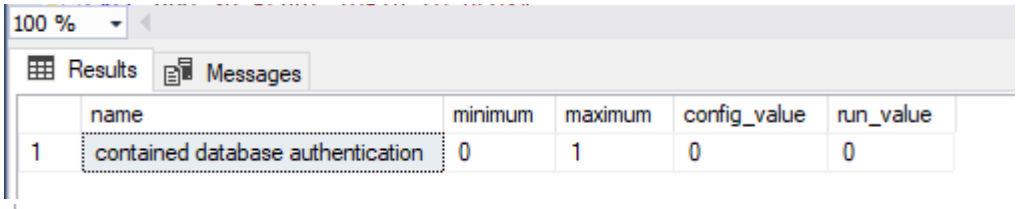
1. Enunciado del ejercicio

   Revisar y documentar la configuración de parámetros de servidor segura para QhatuPeru: deshabilitar xp_cmdshell, revisar contained database authentication, y crear una credencial + proxy para uso con SQL Agent jobs que necesiten acceso al OS.

Script de la solución

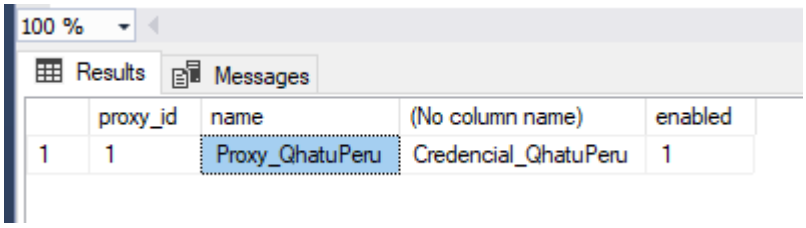
```
-- Deshabilitar xp_cmdshell (si está habilitado)
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;

EXEC sp_configure 'xp_cmdshell', 0;
RECONFIGURE;

-- Verificar estado de contained database authentication
EXEC sp_configure 'contained database authentication';
-- Si se desea deshabilitar:
-- EXEC sp_configure 'contained database authentication', 0;
-- RECONFIGURE;

-- Crear credencial para SQL Agent (requiere usuario de Windows válido)
-- Simulación: usar usuario actual del sistema
CREATE CREDENTIAL [Credencial_QhatuPeru]
WITH IDENTITY = N'lab04-pc11\user 17',
SECRET = N'ContraseñaSegura123!';

-- Crear proxy para SQL Agent Jobs
USE msdb;

EXEC dbo.sp_add_proxy
    @proxy_name = N'Proxy_QhatuPeru',
    @credential_name = N'Credencial_QhatuPeru',
    @enabled = 1;

-- Asignar el proxy al subsistema CmdExec (ID = 1)
EXEC dbo.sp_grant_proxy_to_subsystem
    @proxy_name = N'Proxy_QhatuPeru',
    @subsystem_id = 1; -- CmdExec

-- Asignar el proxy a un rol de SQL Agent
EXEC dbo.sp_grant_login_to_proxy
    @proxy_name = N'Proxy_QhatuPeru',
    @login_name = N'lab04-pc11\user 17';
```

Resultado



	name	minimum	maximum	config_value	run_value
1	contained database authentication	0	1	0	0

```
SELECT proxy_id, name, N'Credencial_QhatuPeru', enabled
FROM msdb.dbo.sysproxies
WHERE name = 'Proxy_QhatuPeru';
```



Justificación de la técnica aplicada

- Xp_cmdshell permite ejecutar comandos del sistema desde SQL, lo cual representa un riesgo si no se controla
- Las credenciales permiten a SQL Server usar una identidad de windows para tareas externas
- El Proxy vincula esa credencial a SQL Agent Jobs, lo que permite que se realicen tareas como CmdExec sin exponer directamente al usuario

Explicación de las buenas prácticas utilizadas

- Desactivamos por defecto funciones inseguras como xp_cmdshell
- Revisamos los parámetros avanzados con sp_configure
- Usamos credenciales y proxies para separar privilegios entre ejecución de tareas y acceso a datos

Enunciado 3

Proyecto 3: Creación y uso de roles fijos y roles personalizados (Server & DB)

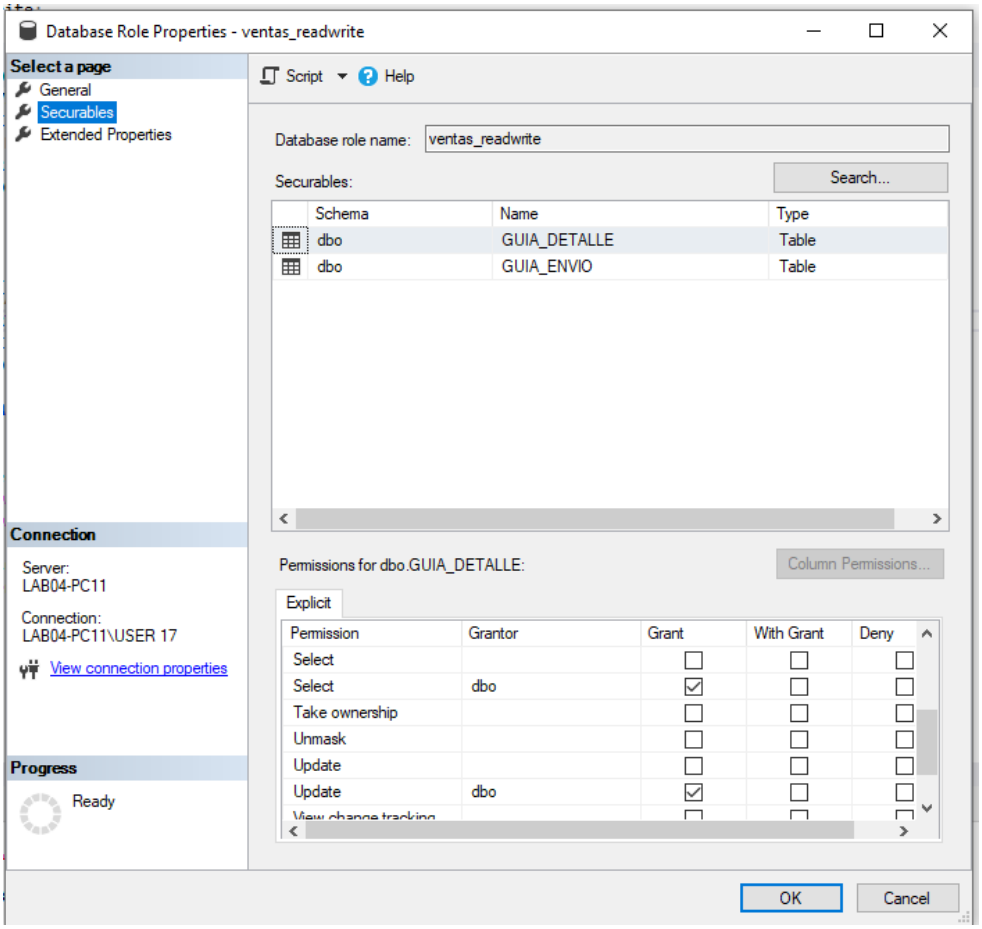
1. Enunciado del ejercicio

   Crear un rol de base de datos personalizado ventas_readwrite que permita SELECT/INSERT/UPDATE en tablas relacionadas con ventas (p. ej. GUIA_ENVIO, GUIA_DETALLE) y asignar usuarios. Mostrar diferencias con roles fijos como db_datareader.

Script de la solución

```sql
USE QhatuPeru;

-- Crear rol personalizado
CREATE ROLE ventas_readwrite;

--tablas de ejemplo
-- Crear tabla GUIA_ENVIO
CREATE TABLE dbo.GUIA_ENVIO (
    ID_GUIA INT PRIMARY KEY IDENTITY(1,1),
    FECHA_ENVIO DATE NOT NULL,
    DESTINATARIO NVARCHAR(100),
    DIRECCION NVARCHAR(200),
    ESTADO NVARCHAR(50)
);

-- Crear tabla GUIA_DETALLE
CREATE TABLE dbo.GUIA_DETALLE (
    ID_DETALLE INT PRIMARY KEY IDENTITY(1,1),
    ID_GUIA INT FOREIGN KEY REFERENCES dbo.GUIA_ENVIO(ID_GUIA),
    PRODUCTO NVARCHAR(100),
    CANTIDAD INT,
    PRECIO_UNITARIO DECIMAL(10,2)
);

-- Conceder permisos específicos sobre tablas de ventas
GRANT SELECT, INSERT, UPDATE ON dbo.GUIA_ENVIO TO ventas_readwrite;
GRANT SELECT, INSERT, UPDATE ON dbo.GUIA_DETALLE TO ventas_readwrite;

-- Asignar el rol a un usuario existente (ejemplo: usuario_sql_alumno)
EXEC sp_addrolemember 'ventas_readwrite', 'usuario_sql_alumno';
```

Resultado



Justificación de la técnica aplicada

- Usamos CREATE ROLE para definir roles personalizados con permisos especificos
- GRAN SELECT , INSERT , UPDATE solo sobre las tablas concretas
- Sp_addrolemember vincula usuarios a roles, lo que facilita la administración de permisos
- El rol db_reader es un rol fijo que solo tiene acceso a lectura de las tablas

Diferencias entre roles fijos y personalizados

Característica	Rol fijo `db_datareader`	Rol personalizado `ventas_readwrite`
Tipo	Predefinido por SQL Server	Definido por el usuario
Permisos	SELECT en todas las tablas	SELECT/INSERT/UPDATE en tablas específicas
Flexibilidad	Limitada	Alta
Riesgo de sobreacceso	Alto	Bajo
Ideal para	Lectura general	Operaciones controladas en áreas clave

Explicación de las buenas prácticas utilizadas

- Usamos los roles personalizados desde el inicio con el principio de mínimos privilegios
- Separamos las funciones entre usuarios con accesos de lectura global y los con acceso operativo limitado
- Aplicamos los permisos explícitos sobre tablas criticas evitando accesos innecesarios

Enunciado 4

Proyecto 4: Control de acceso con GRANT / DENY / REVOKE

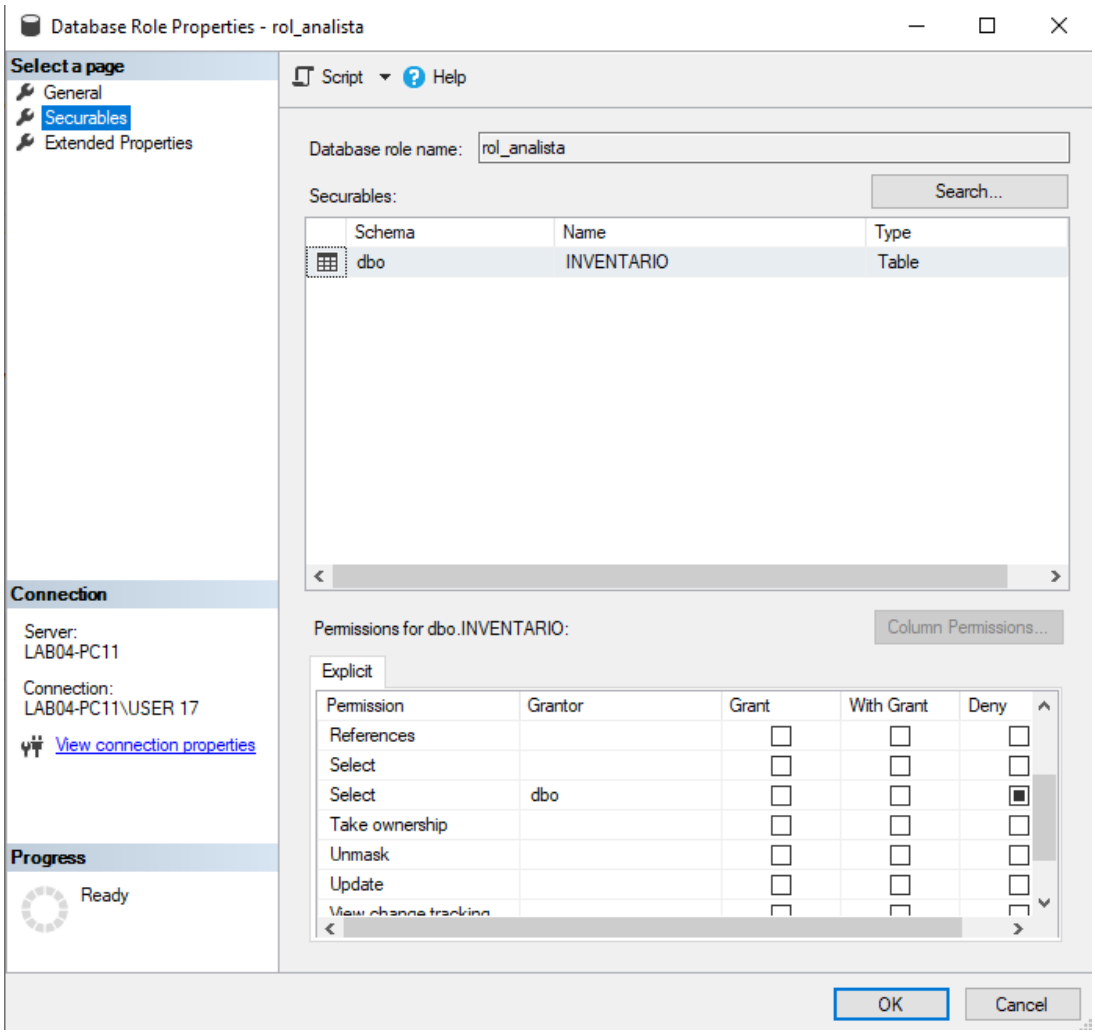
1. Enunciado del ejercicio

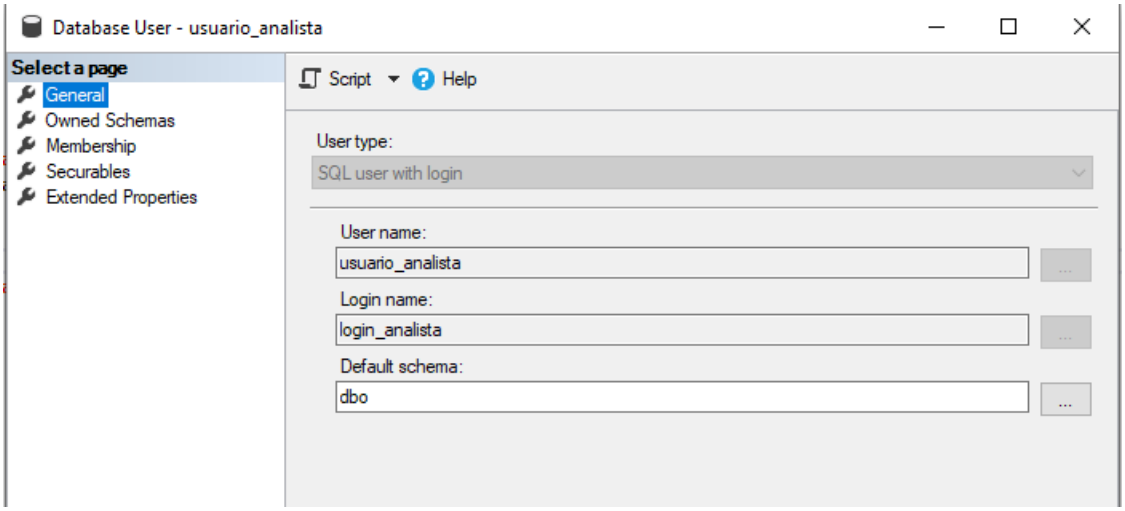
   Simular un caso donde un analista necesita ver inventario pero no los precios. Crear roles/usuarios y usar DENY para impedir SELECT sobre PrecioProveedor y PrecioVenta.

Script de la solución

```sql
-- tabla con precios
USE QhatuPeru;

CREATE TABLE dbo.INVENTARIO (
    ID_PRODUCTO INT PRIMARY KEY IDENTITY(1,1),
    NOMBRE NVARCHAR(100),
    STOCK INT,
    PrecioProveedor DECIMAL(10,2),
    PrecioVenta DECIMAL(10,2)
);

-- Crear login y usuario para el analista
CREATE LOGIN login_analista WITH PASSWORD = 'AnalistaSeguro2024!';
CREATE USER usuario_analista FOR LOGIN login_analista;

-- Crear rol de analista
CREATE ROLE rol_analista;
EXEC sp_addrolemember 'rol_analista', 'usuario_analista';

-- Conceder SELECT general sobre la tabla
GRANT SELECT ON dbo.INVENTARIO TO rol_analista;

-- Negar acceso a columnas sensibles
DENY SELECT ON dbo.INVENTARIO (PrecioProveedor, PrecioVenta) TO rol_analista;
```

Resultado





Justificación de la técnica aplicada

- Con GRANT SELECT se permite al analista consultar la tabla inventario
- DENY SELECT sobre columnas que especificamos le impide ver precios, incluso aunque el usuario tenga permisos generales
- DENY es prioridad sobre GRANT, lo que garantiza columnas protegidas

Explicación de las buenas prácticas utilizadas

- El uso de DENY para proteger datos sensibles de las tablas es muy importante
- Separación de permisos por tablas y columnas
- Creación de roles para facilitar administración de seguridad

Enunciado 5

Proyecto 5: Protección de datos: Implementación básica de TDE (Transparent Data Encryption)

1. Enunciado del ejercicio

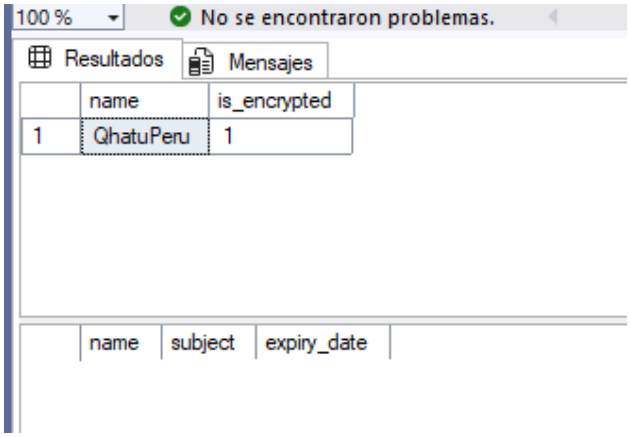
   Habilitar TDE en la base QhatuPeru para proteger los archivos MDF/LDF en reposo. Crear la master key, el certificado de servidor y activar el cifrado.

Script de la solución

```
-- Paso 1: Crear master key en la base master
USE master;
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'ClaveSeguraMaster2024!';

-- Paso 2: Crear certificado de servidor
CREATE CERTIFICATE Certificado_QhatuPeru
WITH SUBJECT = 'Certificado para TDE en QhatuPeru';

-- Paso 3: Crear la Database Encryption Key en la base QhatuPeru
USE QhatuPeru;
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE Certificado_QhatuPeru;

-- Paso 4: Activar el cifrado
ALTER DATABASE QhatuPeru
SET ENCRYPTION ON;

--
-- Verificar bases cifradas
SELECT name, is_encrypted
FROM sys.databases
WHERE name = 'QhatuPeru';

-- Verificar certificados
SELECT name, subject, expiry_date
FROM sys.certificates
WHERE name = 'Certificado_QhatuPeru';
```

Resultado

```
-- Paso 1: Crear master key en la base master
USE master;
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'ClaveSeguraMaster2024!';

-- Paso 2: Crear certificado de servidor
CREATE CERTIFICATE Certificado_QhatuPeru
WITH SUBJECT = 'Certificado para TDE en QhatuPeru';

-- Paso 3: Crear la Database Encryption Key en la base QhatuPeru
USE QhatuPeru;
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE Certificado_QhatuPeru;

-- Paso 4: Activar el cifrado
ALTER DATABASE QhatuPeru
SET ENCRYPTION ON;

--
-- Verificar bases cifradas
SELECT name, is_encrypted
FROM sys.databases
WHERE name = 'QhatuPeru';

-- Verificar certificados
SELECT name, subject, expiry_date
FROM sys.certificates
WHERE name = 'Certificado_QhatuPeru';
```



Justificación de la técnica aplicada

- La master key protege objetos criptográficos en la base MASTER
- El certificado de servidor se usa para cifrar la clave de la base de datos
- La Database Encryption key (DEK) es la clave principal que cifra los datos
- Al activar TDE, todos los archivos físicos (MDF, LDF, BAK) quedan cifrados automáticamente en disco.

Explicación de las buenas prácticas utilizadas

- Uso del algoritmo fuerte (AES_256) para máxima seguridad.
- Separación clara entre master key, certificado y DEK.
- Validación explícita del estado de cifrado.
- Protección de datos en reposo, cumpliendo estándares de seguridad académicos y empresariales.

Enunciado 6

Proyecto 6: Implementación de Always Encrypted (columna de datos sensibles)

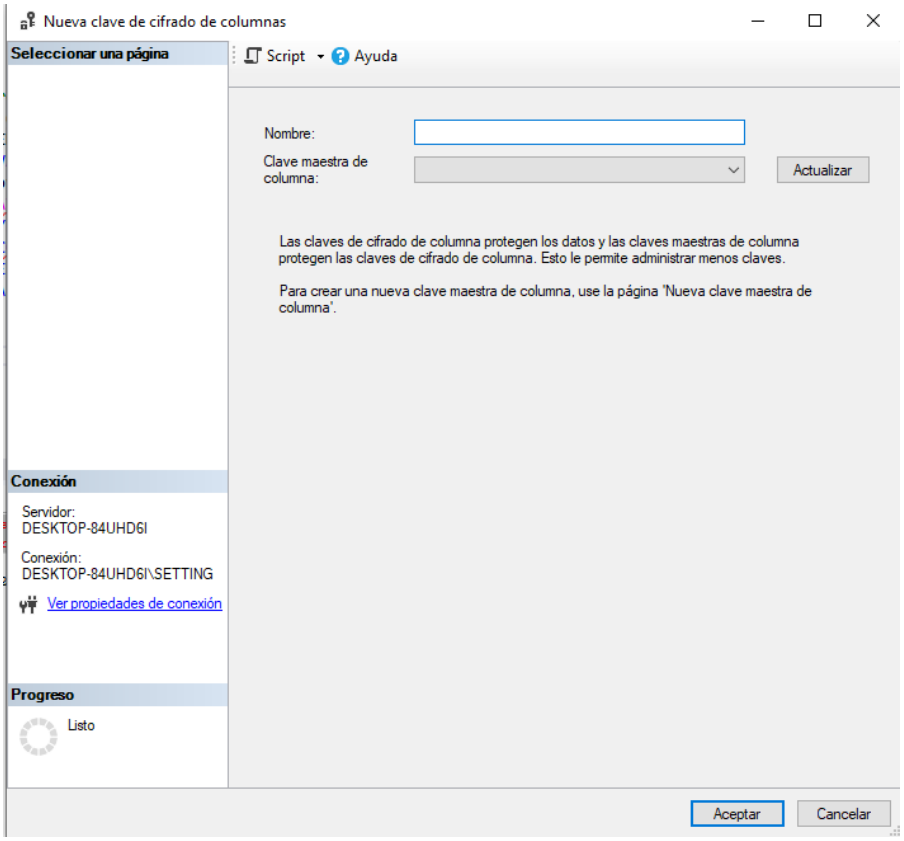
1. Enunciado del ejercicio

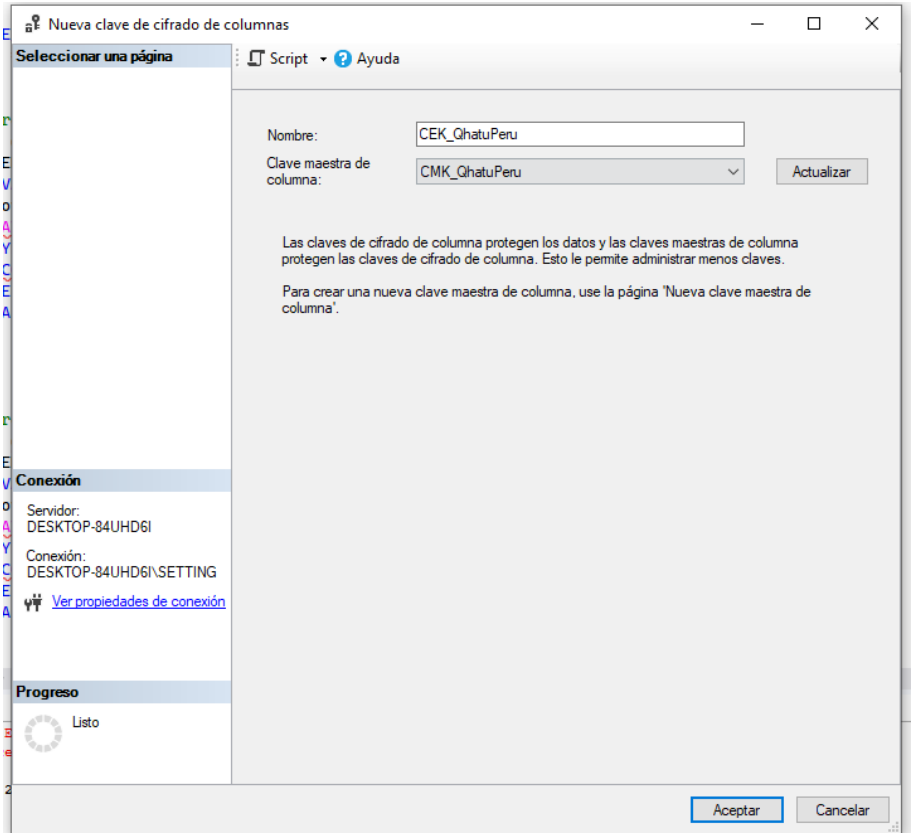
   Configurar un ejemplo de **Always Encrypted** para la columna PrecioProveedor (o crear una nueva columna PrecioProveedor_ENC) usando una Column Master Key (CMK) almacenada en el almacén de certificados y una Column Encryption Key (CEK). Mostrar DDL que crea la columna cifrada

Script de la solución

```sql
-- Paso 1: Crear Column Master Key (CMK)
CREATE COLUMN MASTER KEY CMK_QhatuPeru
WITH (
    KEY_STORE_PROVIDER_NAME = N'MSSQL_CERTIFICATE_STORE',
    KEY_PATH = N'CurrentUser/my/CertificadoAlwaysEncrypted'
);

-- Paso 2: Crear Column Encryption Key (CEK)
CREATE TABLE dbo.PROVEEDORES (
    ID_PROVEEDOR INT PRIMARY KEY IDENTITY(1,1),
    NOMBRE NVARCHAR(100),
    PrecioProveedor_ENC DECIMAL(10,2)
        COLLATE Latin1_General_BIN2
        ENCRYPTED WITH (
            COLUMN_ENCRYPTION_KEY = CEK_QhatuPeru,
            ENCRYPTION_TYPE = Randomized,
            ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'
        )
);

-- Paso 3: Crear tabla con columna cifrada
CREATE TABLE dbo.PROVEEDORES (
    ID_PROVEEDOR INT PRIMARY KEY IDENTITY(1,1),
    NOMBRE NVARCHAR(100),
    PrecioProveedor_ENC DECIMAL(10,2)
        COLLATE Latin1_General_BIN2
        ENCRYPTED WITH (
            COLUMN_ENCRYPTION_KEY = CEK_QhatuPeru,
            ENCRYPTION_TYPE = Randomized,
            ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'
        )
);
```

Resultado





Justificación de la técnica aplicada

- La **CMK** se almacena en el sistema operativo (Windows Certificate Store) y protege las claves de cifrado de columnas.
- La **CEK** se usa para cifrar los datos de la columna en la base de datos.
- La columna `PrecioProveedor_ENC` queda cifrada en disco y en memoria del servidor; solo el cliente con acceso al certificado puede leerla.
- Se usa `Randomized` como tipo de cifrado para mayor seguridad (no permite búsquedas directas, pero evita patrones repetidos).

Explicación de las buenas prácticas utilizadas

- Separación entre CMK y CEK para mayor seguridad.
- Uso de algoritmos fuertes (`AEAD_AES_256_CBC_HMAC_SHA_256`).
- Protección de datos sensibles a nivel de columna, incluso frente a administradores.

Enunciado 7

Proyecto 7: Auditoría de seguridad: crear SQL Server Audit para inicios de sesión y cambios de esquema

1. Enunciado del ejercicio

   Configurar un Server Audit que registre intentos de login fallidos y exitosos, y un Database Audit Specification que registre cambios DDL en QhatuPeru (CREATE/ALTER/DROP para objetos críticos).

Script de la solución

```
USE master;
--paso 1 creamos el server audit
-- Crear el audit en el servidor, con archivo de salida
CREATE SERVER AUDIT Audit_QhatuPeru
TO FILE (FILEPATH = 'E:\SQLAudit\')
WITH (ON_FAILURE = CONTINUE);

-- Habilitar el audit
ALTER SERVER AUDIT Audit_QhatuPeru
WITH (STATE = ON);

--paso 2
-- Registrar intentos de login
CREATE SERVER AUDIT SPECIFICATION Audit_Logins
FOR SERVER AUDIT Audit_QhatuPeru
ADD (SUCCESSFUL_LOGIN_GROUP),
ADD (FAILED_LOGIN_GROUP)
WITH (STATE = ON);

--paso 3
-- crear la database para cambios DDL
USE QhatuPeru;

CREATE DATABASE AUDIT SPECIFICATION Audit_DDL_QhatuPeru
FOR SERVER AUDIT Audit_QhatuPeru
ADD (SCHEMA_OBJECT_CHANGE_GROUP)
WITH (STATE = ON);
```

Resultado

```
-- verificaciones
-- Auditorías a nivel servidor
SELECT name, audit_guid, is_state_enabled
FROM sys.server_audit_specifications;

-- Auditorías a nivel base de datos
SELECT name, is_state_enabled
FROM sys.database_audit_specifications;
```

100 % 1 0

Resultados Mensajes

	name	audit_guid	is_state_enabled
1	Audit_Logins	F498D3D4-3EDD-49D4-AE26-14610F90A16A	1

	name	is_state_enabled
1	Audit_DDL_QhatuPeru	1

Justificación de la técnica aplicada

- El **Server Audit** define el destino de los registros (archivo en disco).
- El **Server Audit Specification** captura eventos de inicio de sesión exitosos y fallidos.
- El **Database Audit Specification** captura cambios de esquema (DDL) en la base `QhatuPeru`.

Explicación de las buenas prácticas utilizadas

- Separación entre auditoría de logins (nivel servidor) y auditoría de cambios DDL (nivel base de datos).
- Uso de `ON_FAILURE = CONTINUE` para evitar bloqueos en caso de error de escritura.
- Almacenamiento en archivo para trazabilidad y análisis posterior.

Enunciado 8

Proyecto 8: Monitoreo de eventos y alertas con Extended Events + Auditoría

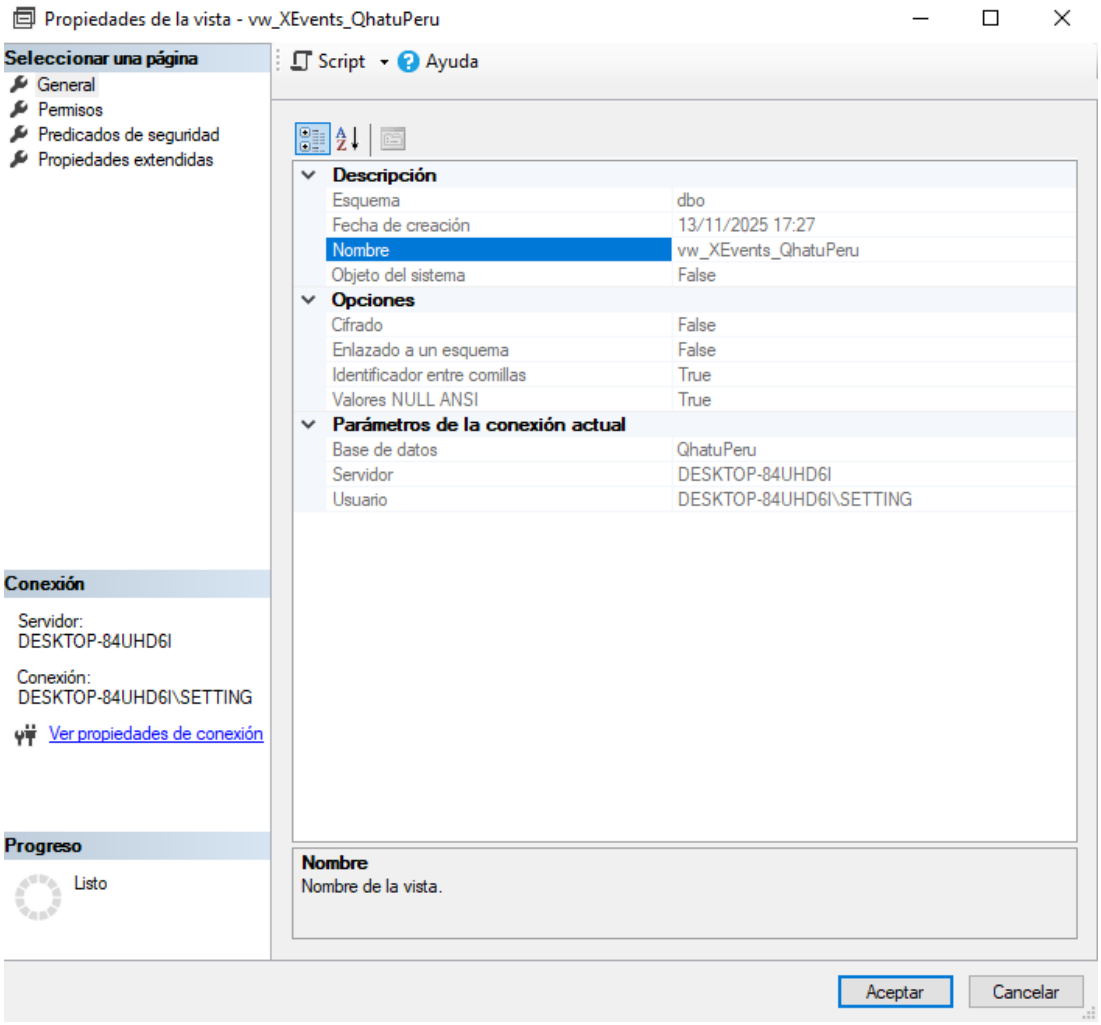
1. Enunciado del ejercicio

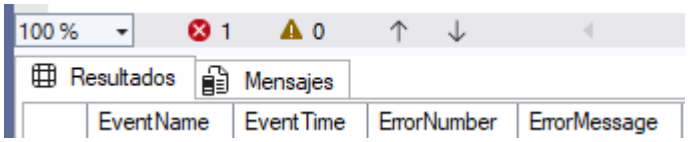
   Configurar una sesión de **Extended Events** que capture deadlocks y eventos de login failed, guardar en archivo y crear una vista que permita consultar los XEvent desde la base.

Script de la solución

```
-- Crear sesión de Extended Events para deadlocks y login failed
CREATE EVENT SESSION Sesion_QhatuPeru
ON SERVER
ADD EVENT sqlserver.xml_deadlock_report,    -- Captura deadlocks en XML
ADD EVENT sqlserver.error_reported(
    WHERE (error_number = 18456)            -- Login failed
)
ADD TARGET package0.event_file (
    SET filename = 'E:\XEvents\Sesion_QhatuPeru.xel',
        max_file_size = 10,
        max_rollover_files = 5
);

-- Iniciar la sesión
ALTER EVENT SESSION Sesion_QhatuPeru ON SERVER STATE = START;

--paso 2

CREATE VIEW vw_XEvents_QhatuPeru
AS
SELECT
    event_data.value('(event/@name)[1]', 'NVARCHAR(100)') AS EventName,
    event_data.value('(event/@timestamp)[1]', 'DATETIME2') AS EventTime,
    event_data.value('(event/data[@name="error_number"]/value)[1]', 'INT') AS ErrorNumber,
    event_data.value('(event/data[@name="message"]/value)[1]', 'NVARCHAR(4000)') AS ErrorMessage
FROM (
    SELECT CAST(event_data AS XML) AS event_data
    FROM sys.fn_xe_file_target_read_file(
        'E:\XEvents\Sesion_QhatuPeru*.xel', NULL, NULL, NULL
    )
) AS XEvents;
```

Resultado





Justificación de la técnica aplicada

- `sqlserver.deadlock_graph` captura información detallada de los procesos involucrados en un deadlock.
- `sqlserver.error_reported` con filtro `error_number = 18456` captura intentos de login fallidos.
- El **target** `event_file` guarda los eventos en disco para análisis posterior.
- La vista `vw_XEvents_QhatuPeru` permite consultar los eventos directamente desde la base, integrando monitoreo con auditoría.

Explicación de las buenas prácticas utilizadas

- Uso de Extended Events en lugar de SQL Trace (más eficiente y moderno).
- Filtrado de eventos para evitar ruido innecesario.

- Almacenamiento en archivo con rotación controlada para no saturar disco.
- Exposición de datos mediante vista para facilitar consultas y reportes.

Enunciado 9

Proyecto 9: Implementación de enmascaramiento dinámico + acceso controlado

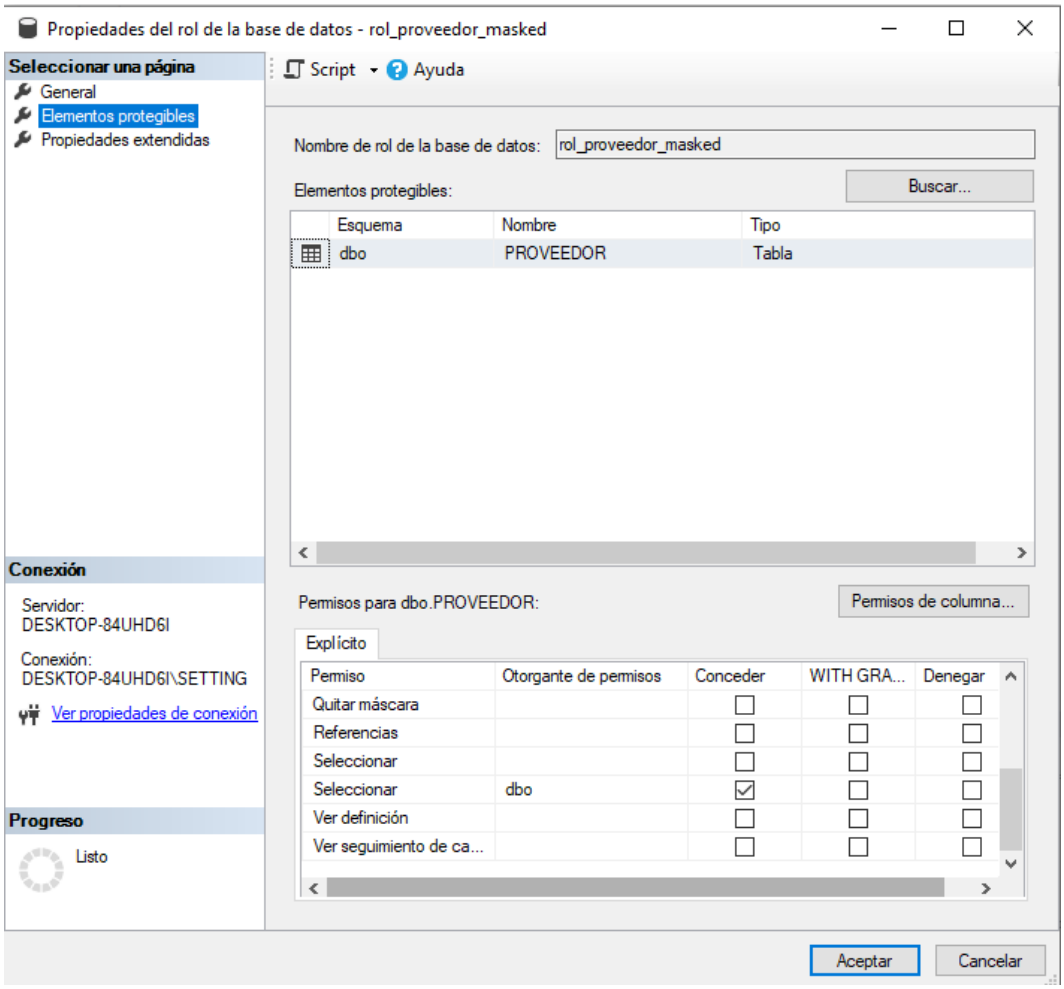
1. Enunciado del ejercicio

   Aplicar Dynamic Data Masking a columnas sensibles (por ejemplo Telefono en PROVEEDOR) y crear una vista segura para usuarios que necesiten ver datos completos mediante una función que valide rol.

Script de la solución

```
USE QhatuPeru;

CREATE TABLE dbo.PROVEEDOR (
    ID_PROVEEDOR INT PRIMARY KEY IDENTITY(1,1),
    NOMBRE NVARCHAR(100),
    Telefono NVARCHAR(20) MASKED WITH (FUNCTION = 'partial(0,"XXX-XXX-",4)')
);

--cremaos rol y un usuario enmascarado
-- Crear login y usuario
CREATE LOGIN login_proveedor WITH PASSWORD = 'ProveedorSeguro2024!';
CREATE USER usuario_proveedor FOR LOGIN login_proveedor;

-- Rol de acceso limitado
CREATE ROLE rol_proveedor_masked;
EXEC sp_addrolemember 'rol_proveedor_masked', 'usuario_proveedor';

-- Conceder permisos de lectura
GRANT SELECT ON dbo.PROVEEDOR TO rol_proveedor_masked;

--creamos funcion para validar rol
-- Función que valida si el usuario pertenece a un rol
CREATE FUNCTION fn_ValidarRol(@Rol NVARCHAR(100))
RETURNS BIT
AS
BEGIN
    DECLARE @Result BIT = 0;
    IF IS_ROLEMEMBER(@Rol) = 1
        SET @Result = 1;
    RETURN @Result;
END;

--vista para acceso compleo
-- Vista que muestra datos completos solo si el usuario pertenece al rol seguro
CREATE VIEW vw_ProveedorSeguro
AS
SELECT
    ID_PROVEEDOR,
    NOMBRE,
    CASE
        WHEN dbo.fn_ValidarRol('db_owner') = 1
        THEN Telefono
        ELSE '***ENMASCARADO***'
    END AS TelefonoSeguro
FROM dbo.PROVEEDOR;
```

Resultado



Justificación de la técnica aplicada

- **Dynamic Data Masking** protege columnas sensibles mostrando valores parciales o ficticios.
- La **función** `fn_ValidarRol` permite condicionar la visibilidad de datos completos según el rol del usuario.
- La **vista segura** centraliza la lógica de acceso, evitando que usuarios sin privilegios vean información real.
- Se aplica el principio de **mínimos privilegios**

Explicación de las buenas prácticas utilizadas

- Uso de DDM para proteger datos sensibles sin modificar la aplicación.
- Validación de roles con funciones para control granular.
- Creación de vistas seguras que encapsulan la lógica de acceso.
- Separación clara entre usuarios con acceso limitado y administradores.

Enunciado 10

Proyecto 10: Capstone: Integración (roles, TDE, Always Encrypted, auditoría)

1. Enunciado del ejercicio

   Proyecto integrador: crear un rol auditor_seguridad, habilitar TDE (si no está), preparar Always Encrypted para columna sensible, configurar auditoría de accesos a esa tabla, y dejar un procedimiento almacenado que registre cambios críticos (traza soportada por audit).

Script de la solución

```sql
USE QhatuPeru;

CREATE ROLE auditor_seguridad;
-- Asignar usuario existente al rol
EXEC sp_addrolemember 'auditor_seguridad', 'usuario_sql_alumno';

-- Crear master key y certificado si no existen
USE master;
IF NOT EXISTS (SELECT * FROM sys.symmetric_keys WHERE name = '##MS_DatabaseMasterKey##')
    CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'ClaveMaster2024!';

IF NOT EXISTS (SELECT * FROM sys.certificates WHERE name = 'Certificado_QhatuPeru')
    CREATE CERTIFICATE Certificado_QhatuPeru
    WITH SUBJECT = 'Certificado para TDE en QhatuPeru';

-- Crear Database Encryption Key y activar TDE
USE QhatuPeru;
IF NOT EXISTS (SELECT * FROM sys.dm_database_encryption_keys WHERE database_id = DB_ID('QhatuPeru'))
BEGIN
    CREATE DATABASE ENCRYPTION KEY
    WITH ALGORITHM = AES_256
    ENCRYPTION BY SERVER CERTIFICATE Certificado_QhatuPeru;

    ALTER DATABASE QhatuPeru SET ENCRYPTION ON;
END

-- Supongamos que ya existe CMK y CEK creados en SSMS
CREATE TABLE dbo.CLIENTES (
    ID_CLIENTE INT PRIMARY KEY IDENTITY(1,1),
    NOMBRE NVARCHAR(100),
    DNI NVARCHAR(20)
        COLLATE Latin1_General_BIN2
        ENCRYPTED WITH (
            COLUMN_ENCRYPTION_KEY = CEK_QhatuPeru,
            ENCRYPTION_TYPE = Randomized,
            ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'
        )
);
```

```sql
USE master;

-- Crear audit si no existe
IF NOT EXISTS (SELECT * FROM sys.server_audits WHERE name = 'Audit_QhatuPeru')
    CREATE SERVER AUDIT Audit_QhatuPeru
    TO FILE (FILEPATH = 'C:\SQLAudit\')
    WITH (ON_FAILURE = CONTINUE);

ALTER SERVER AUDIT Audit_QhatuPeru WITH (STATE = ON);

-- Auditoría a nivel base de datos
USE QhatuPeru;
CREATE DATABASE AUDIT SPECIFICATION Audit_Accesos_QhatuPeru
FOR SERVER AUDIT Audit_QhatuPeru
ADD (SELECT ON dbo.CLIENTES BY auditor_seguridad),
ADD (UPDATE ON dbo.CLIENTES BY auditor_seguridad)
WITH (STATE = ON);

CREATE PROCEDURE sp_ActualizarCliente
    @ID_CLIENTE INT,
    @NuevoNombre NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE dbo.CLIENTES
    SET NOMBRE = @NuevoNombre
    WHERE ID_CLIENTE = @ID_CLIENTE;

    -- Registrar acción en tabla de trazas
    INSERT INTO dbo.TrazaCambios (Usuario, Fecha, Accion, ClienteID)
    VALUES (SYSTEM_USER, GETDATE(), 'UPDATE CLIENTE', @ID_CLIENTE);
END;
```

Justificación de la técnica aplicada

- El rol `auditor_seguridad` centraliza permisos de monitoreo.
- TDE protege archivos físicos en reposo.
- Always Encrypted protege columnas sensibles incluso frente a administradores.
- La auditoría captura accesos y modificaciones en la tabla crítica.
- El procedimiento almacenado asegura trazabilidad interna y externa (auditoría + tabla de trazas).

Explicación de las buenas prácticas utilizadas

- Integración de múltiples capas de seguridad (roles, cifrado, auditoría).
- Principio de mínimos privilegios aplicado a usuarios y roles.
- Documentación clara de cada paso para trazabilidad académica.
- Uso de procedimientos almacenados para controlar y registrar cambios críticos.