S11_Practica

BREÑA QUISPE Megam Briyit

Actividad

Responda con claridad y precisión cada uno de los siguientes ejercicios prácticos.

Proyecto 1: Autenticación: Comparación segura y configuración de logins

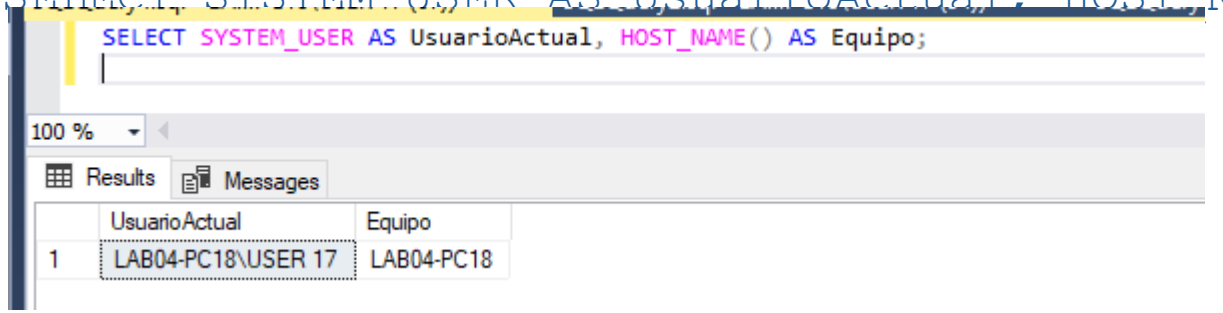
1. Enunciado del ejercicio

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.

2. Script de la solución en T-SQL

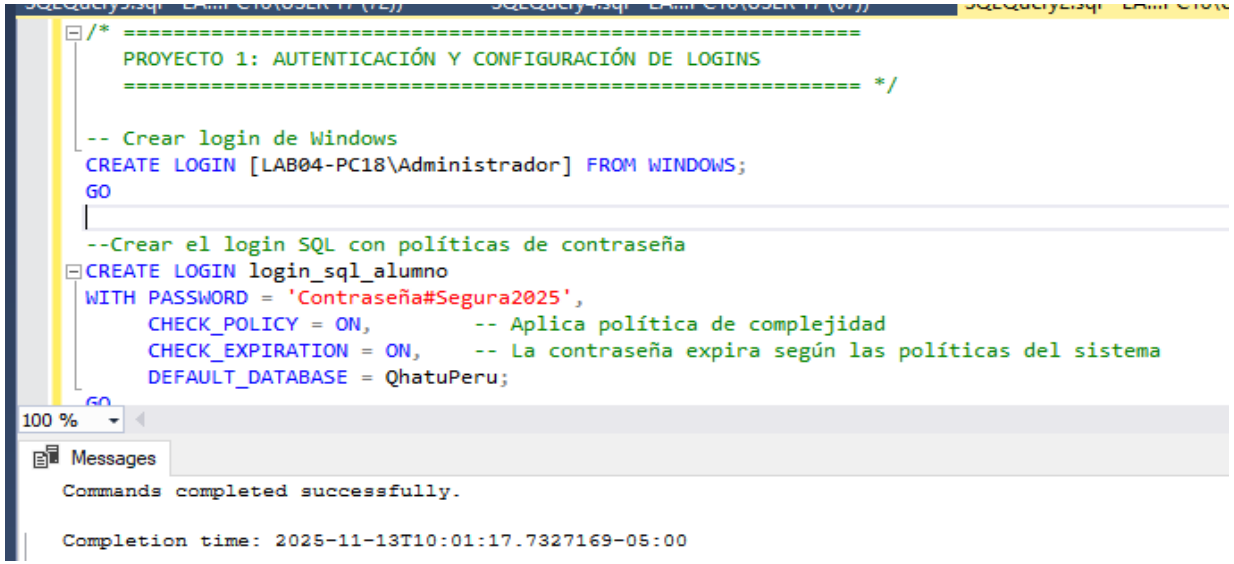
Verifica el contexto del servidor SQL

```
SELECT SYSTEM_USER AS UsuarioActual, HOST_NAME() AS Equipo;
```



Crea un login de Windows válido

```
CREATE LOGIN [lab04-pc18\user 1] FROM WINDOWS;
GO
```



Mapear ambos logins (Windows y SQL) a usuarios dentro de la base

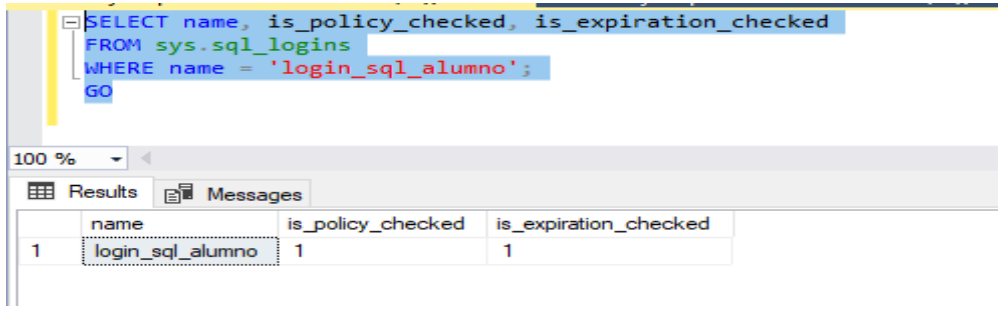
```
USE QhatuPeru;
GO
-- Usuario mapeado al login SQL
CREATE USER usuario_sql_alumno
FOR LOGIN login_sql_alumno;
GO
-- Usuario mapeado al login de Windows
CREATE USER usuario_win_alumno
FOR LOGIN [LAB04-PC18\user17];
GO
```

Crear el login SQL con políticas de contraseña

```
CREATE LOGIN login_sql_alumno
WITH PASSWORD =
'Contraseña#Segura2025',
     CHECK_POLICY =
ON,        -- Aplica política
de complejidad
     CHECK_EXPIRATION =
ON,    -- La contraseña expira
según las políticas del
sistema
     DEFAULT_DATABASE =
QhatuPeru;
GO
```

Comprobar la política de contraseñas

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



3. Justificación técnica de la solución aplicada

- **Creación de logins diferenciados según tipo de autenticación:** Se creó un login de SQL Server (**login_sql_alumno**) con contraseña y políticas de seguridad activadas.
- Se creó un login de Windows (**LAB04-PC18\alumno_win**) para simular la integración con autenticación de sistema operativo o dominio.
  Esto permite separar claramente los mecanismos de autenticación y garantiza que cada tipo de usuario se gestione según su contexto.
- **Aplicación de políticas de contraseña y expiración**
- La contraseña del login SQL se configuró con **CHECK_POLICY = ON** y **CHECK_EXPIRATION = ON.**
- Se forzó la expiración con **MUST_CHANGE** para probar el ciclo de seguridad de contraseñas.
  Esto asegura que las contraseñas cumplan con la complejidad mínima y que se renueven periódicamente, reduciendo riesgos de seguridad.

4. Explicación de las buenas prácticas utilizadas en el proyecto

Separación de logins y usuarios de base de datos

- Mantener la autenticación a nivel de servidor (`LOGIN`) y el acceso a nivel de base (`USER`) separados permite controlar permisos de manera granular y segura.

Uso de políticas de contraseña

- `CHECK_POLICY` y `CHECK_EXPIRATION` garantizan que las contraseñas cumplan con estándares de seguridad y se renueven periódicamente.
- `MUST_CHANGE` permite simular y probar la expiración obligatoria, fomentando hábitos seguros de gestión de contraseñas.

Validación de existencia antes de crear objetos

- Se utilizaron `IF EXISTS` y `IF NOT EXISTS` para evitar errores de creación de usuarios o logins que ya existen.
- Esto asegura que los scripts sean **idempotentes** y puedan ejecutarse varias veces sin fallar.

Simulación de usuarios de Windows

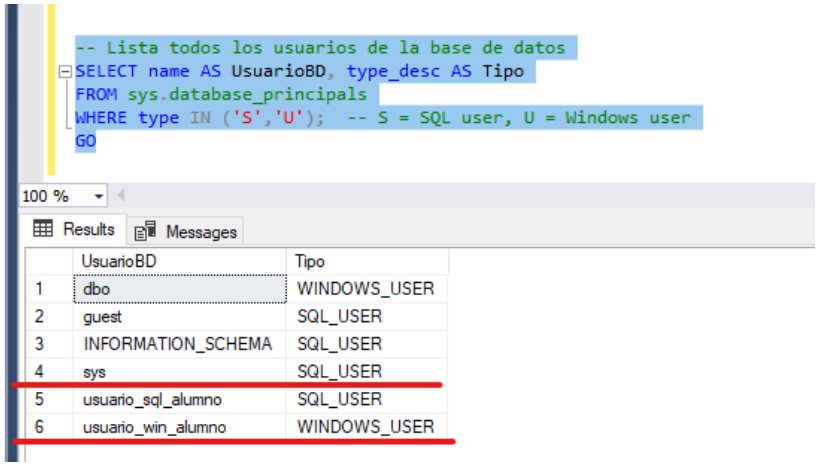
- Usar un login de Windows real o simulado permite practicar escenarios de integración con dominios y autenticación externa, acercando la práctica a entornos reales.

Documentación y trazabilidad

- Cada paso del script está comentado y justificado, lo que facilita mantenimiento y auditoría de seguridad.

Resultado Esperado:

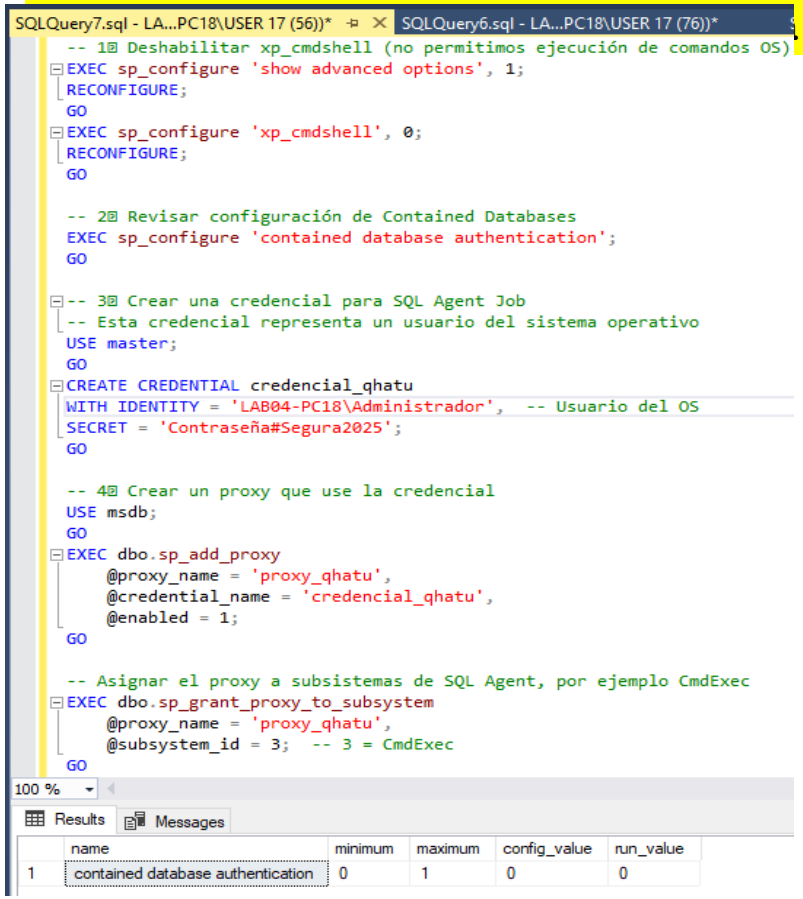
```
USE QhatuPeru;
GO
-- Lista todos los usuarios de la
base de datos
SELECT name AS UsuarioBD, type_desc
AS Tipo
FROM sys.database_principals
WHERE type IN ('S','U');  -- S =
SQL user, U = Windows user
```



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.



T-SQL

```
-- 1 Deshabilitar xp_cmdshell (no permitimos ejecución de comandos OS)
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
GO
EXEC sp_configure 'xp_cmdshell', 0;
RECONFIGURE;
GO
-- 2 Revisar configuración de Contained Databases
EXEC sp_configure 'contained database authentication';
GO
-- 3 Crear una credencial para SQL Agent Job
-- Esta credencial representa un usuario del sistema operativo
USE master;
GO
CREATE CREDENTIAL credencial_qhatu
WITH IDENTITY = 'LAB04-PC18\usuario_job',  -- Usuario del OS
SECRET = 'Contraseña#Segura2025';
GO
-- 4 Crear un proxy que use la credencial
USE msdb;
GO
EXEC dbo.sp_add_proxy
    @proxy_name = 'proxy_qhatu',
    @credential_name = 'credencial_qhatu',
    @enabled = 1;
GO
-- Asignar el proxy a subsistemas de SQL Agent, por ejemplo CmdExec
EXEC dbo.sp_grant_proxy_to_subsystem
    @proxy_name = 'proxy_qhatu',
    @subsystem_id = 3;   -- 3 = CmdExec
GO
```

3. Justificación técnica de la solución aplicada

Deshabilitar xp_cmdshell

- Evita que un usuario con permisos pueda ejecutar comandos del sistema operativo desde SQL Server, reduciendo riesgos de ataques o ejecución de malware.

Revisión de Contained Database Authentication

- Permite determinar si la base puede autenticarse de forma independiente del servidor.
- Mejora la portabilidad y seguridad al controlar si se permiten logins internos en la base.

Creación de credencial y proxy

- La credencial representa un **usuario del OS** con permisos limitados.
- El proxy permite que **SQL Agent Jobs** ejecuten tareas de sistema sin usar cuentas administrativas, reduciendo el riesgo de elevación de privilegios.

4. Explicación de las buenas prácticas utilizadas en el proyecto

Principio de mínimo privilegio

- La credencial y el proxy usan un usuario limitado, evitando usar cuentas de administrador para tareas automáticas.

Seguridad de configuración del servidor

- `xp_cmdshell` deshabilitado y revisión de `contained database authentication` para reducir la superficie de ataque.

Separación de responsabilidades

- SQL Agent Jobs usan proxies y credenciales específicas, evitando mezclar usuarios de aplicación, administrador y sistema operativo.

Documentación y trazabilidad

- Cada paso está comentado y justificado para auditoría y mantenimiento.

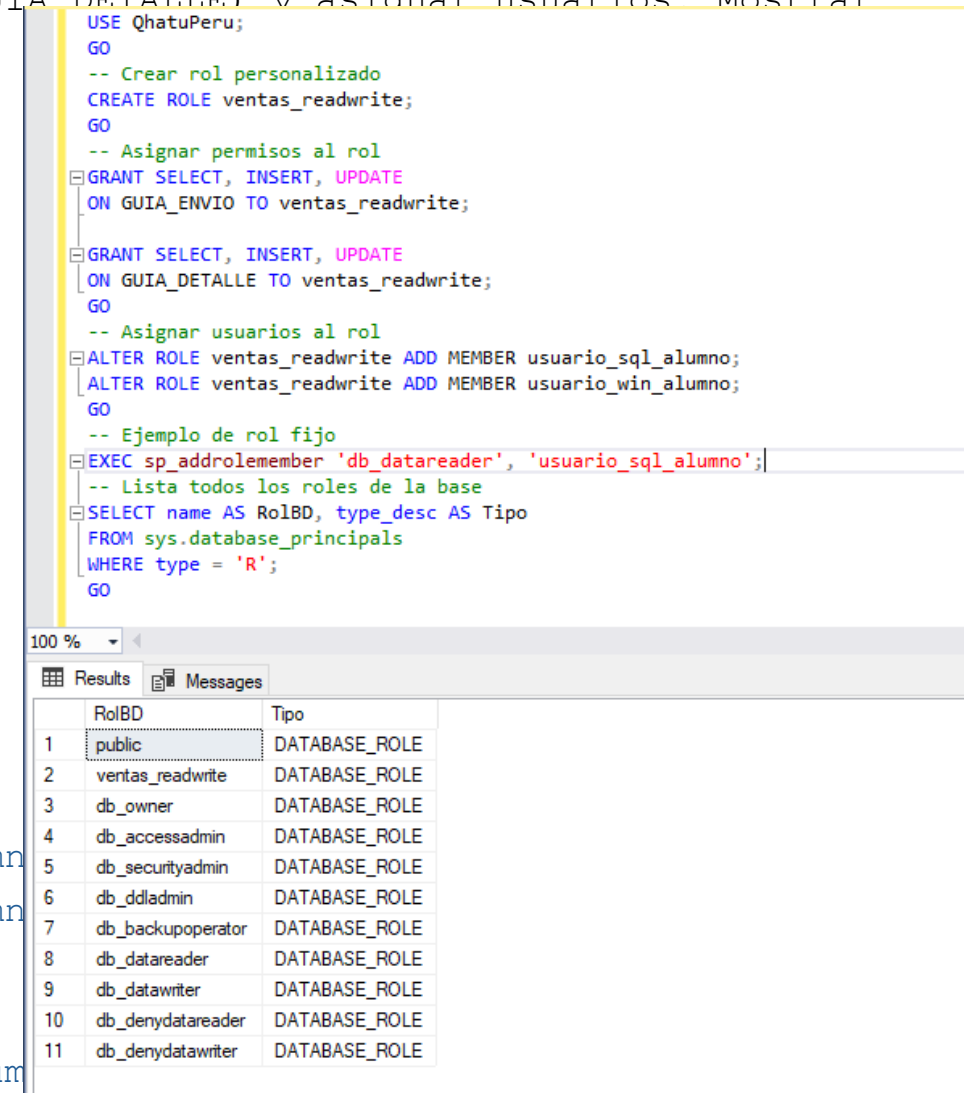
Proyecto 3: Creación y uso de roles fijos y roles personalizados (Server & DB)

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.

2. Script de la solución en T-SQL

```
USE QhatuPeru;
GO
-- Crear rol personalizado
CREATE ROLE ventas_readwrite;
GO
-- Asignar permisos al rol
GRANT SELECT, INSERT, UPDATE
ON GUIA_ENVIO TO ventas_readwrite;

GRANT SELECT, INSERT, UPDATE
ON GUIA_DETALLE TO ventas_readwrite;
GO
-- Asignar usuarios al rol
ALTER ROLE ventas_readwrite ADD MEMBER usuario_sql_alumn
ALTER ROLE ventas_readwrite ADD MEMBER usuario_win_alumn
GO
-- Ejemplo de rol fijo
EXEC sp_addrolemember 'db_datareader', 'usuario_sql_alum
```



```
USE QhatuPeru;
GO
-- Crear rol personalizado
CREATE ROLE ventas_readwrite;
GO
-- Asignar permisos al rol
GRANT SELECT, INSERT, UPDATE
ON GUIA_ENVIO TO ventas_readwrite;

GRANT SELECT, INSERT, UPDATE
ON GUIA_DETALLE TO ventas_readwrite;
GO
-- Asignar usuarios al rol
ALTER ROLE ventas_readwrite ADD MEMBER usuario_sql_alumno;
ALTER ROLE ventas_readwrite ADD MEMBER usuario_win_alumno;
GO
-- Ejemplo de rol fijo
EXEC sp_addrolemember 'db_datareader', 'usuario_sql_alumno';
-- Lista todos los roles de la base
SELECT name AS RolBD, type_desc AS Tipo
FROM sys.database_principals
WHERE type = 'R';
GO
```

100 %

Results | Messages

	RolBD	Tipo
1	public	DATABASE_ROLE
2	ventas_readwrite	DATABASE_ROLE
3	db_owner	DATABASE_ROLE
4	db_accessadmin	DATABASE_ROLE
5	db_securityadmin	DATABASE_ROLE
6	db_ddladmin	DATABASE_ROLE
7	db_backupoperator	DATABASE_ROLE
8	db_datareader	DATABASE_ROLE
9	db_datawriter	DATABASE_ROLE
10	db_denydatareader	DATABASE_ROLE
11	db_denydatawriter	DATABASE_ROLE

3. Justificación técnica de la solución aplicada

- Permite controlar permisos **por grupo**, evitando asignar privilegios uno por uno.
- Los roles fijos (db_datareader) dan permisos generales, mientras los personalizados permiten **control fino por tabla o esquema.**

4. Explicación de las buenas prácticas utilizadas en el proyecto

- Uso de roles para **principio de mínimo privilegio.**
- Separar roles por tipo de acción y tabla para **mejor trazabilidad.**

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 Precio Proveedor y PrecioVenta.

2. Script de la solución en T-SQL

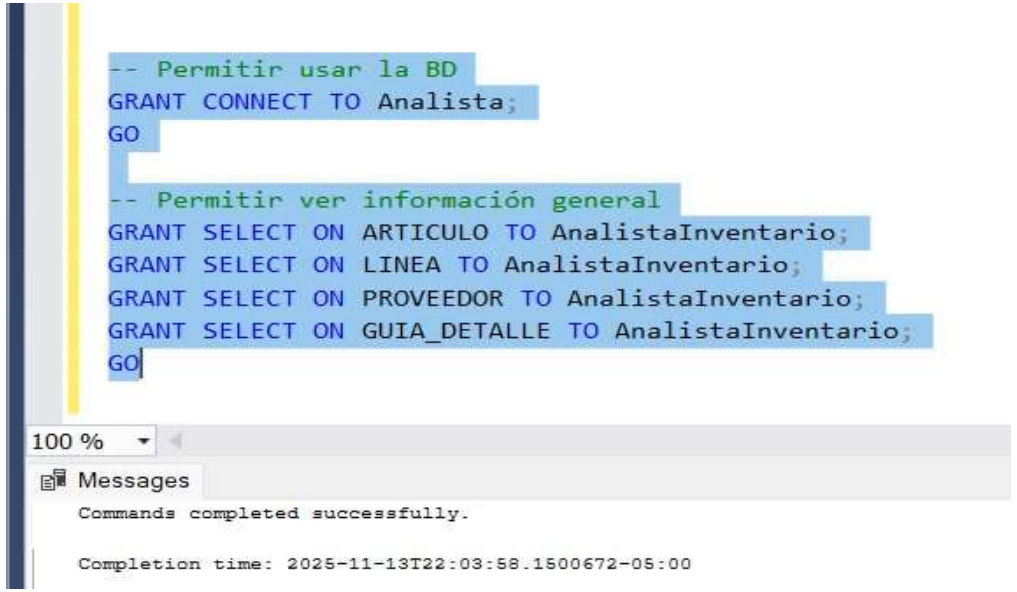
PASO 1 — Crear usuarios y roles

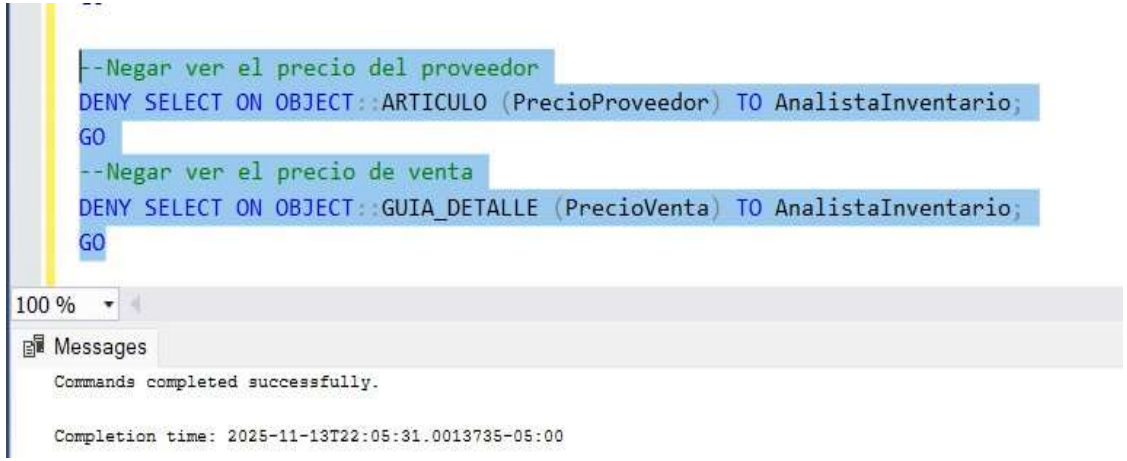
```
USE QhatuPERU;

GO

-- Crear usuario sin login (para pruebas
   locales)

CREATE LOGIN AnalistaLogin WITH PASSWORD =
   'Analista123$', CHECK_POLICY = OFF;

GO

CREATE USER Analista FOR LOGIN AnalistaLogin;

GO

-- Crear rol

CREATE ROLE AnalistaInventario;

GO

-- Agregar usuario al rol

ALTER ROLE AnalistaInventario ADD MEMBER
   Analista;

GO
```

```
USE QhatuPERU;
GO

-- Crear usuario sin login (para pruebas locales)
CREATE LOGIN AnalistaLogin WITH PASSWORD = 'Analista123$', CHECK_POLICY = OFF;
GO

CREATE USER Analista FOR LOGIN AnalistaLogin;
GO

-- Crear rol
CREATE ROLE AnalistaInventario;
GO

-- Agregar usuario al rol
ALTER ROLE AnalistaInventario ADD MEMBER Analista;
GO
```

PASO 2 — Otorgar permisos de consulta sobre tablas necesarias

```
-- Permitir usar la BD
GRANT CONNECT TO Analista;
GO
-- Permitir ver información general
GRANT SELECT ON ARTICULO TO AnalistaInventario;
GRANT SELECT ON LINEA TO AnalistaInventario;
GRANT SELECT ON PROVEEDOR TO AnalistaInventario;
GRANT SELECT ON GUIA_DETALLE TO AnalistaInventario;
```



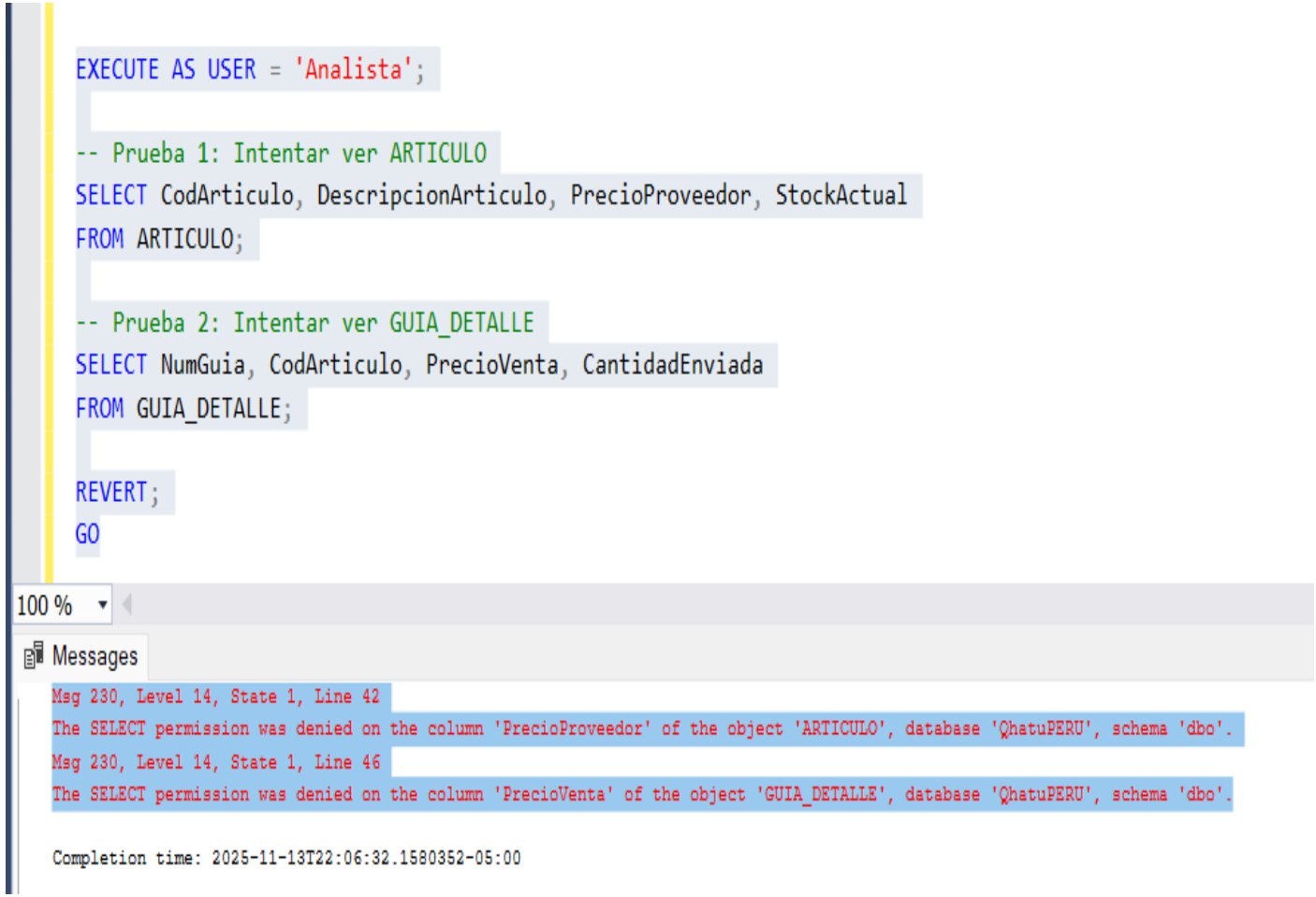


PASO 3 — Aplicar DENY sobre columnas sensibles

```
Negar ver el precio del proveedor
DENY SELECT ON OBJECT::ARTICULO (PrecioProveedor) TO AnalistaInventario;
GO
Negar ver el precio de venta
DENY SELECT ON OBJECT::GUIA_DETALLE (PrecioVenta) TO AnalistaInventario;
GO
```

PASO 4 — Prueba del ejercicio

```
EXECUTE AS USER = 'Analista';

-- Prueba 1: Intentar ver ARTICULO
SELECT CodArticulo,
DescripcionArticulo,
PrecioProveedor, StockActual
FROM ARTICULO;

-- Prueba 2: Intentar ver
GUIA_DETALLE
SELECT NumGuia, CodArticulo,
PrecioVenta, CantidadEnviada
FROM GUIA_DETALLE;

REVERT;
GO
```



Ese mensaje no es un error de mal funcionamiento, sino la confirmación correcta de que el usuario NO puede ver esos datos.

3. Justificación técnica de la solución aplicada

- Principio de privilegios mínimos
  El Analista solo recibe permisos estrictamente necesarios (SELECT sobre tablas específicas).
  No recibe permisos innecesarios como INSERT, UPDATE ni DELETE.
- Uso de roles en lugar de permisos directos
  Facilita la administración: nuevos usuarios solo se agregan al rol.
- DENY explícito sobre columnas específicas
  Aunque un rol tenga SELECT sobre una tabla, un DENY siempre tiene mayor prioridad.
  Esto permite que el usuario consulte inventario pero sin acceder a información sensible.
- Separación de permisos por objeto
- ARTICULO → restricciones en PrecioProveedor
- GUIA_DETALLE → restricciones en PrecioVenta
- Prueba mediante EXECUTE AS USER
  Permite verificar los permisos sin necesidad de cerrar sesión.

4. Explicación de las buenas prácticas utilizadas en el proyecto

Uso del principio de privilegios mínimos

Se dio acceso solo a la información estrictamente necesaria para el rol.

Seguridad basada en roles (RBAC)

En vez de asignar permisos uno por uno, se crea un rol que agrupa permisos de manera profesional.

Separación clara entre datos públicos y datos sensibles

Se protege información financiera: precios de compra y venta.

Uso de DENY solo en casos especiales

Se usa únicamente para bloquear columnas sensibles que están dentro de tablas que el usuario sí necesita consultar.

Auditoría de permisos mediante pruebas reales

Se incluyó un bloque `EXECUTE AS USER` para validar la solución.

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.

2. Script de la solución en T-SQL

```
USE master;
GO
-- Crear master key
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Contraseña#Maestra2025';
GO
-- Crear certificado
CREATE CERTIFICATE TDE_Cert
WITH SUBJECT = 'Certificado para TDE QhatuPeru';
GO
-- Crear algoritmo de cifrado
USE QhatuPeru;
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDE_Cert;
GO
-- Activar TDE
ALTER DATABASE QhatuPeru SET ENCRYPTION ON;
GO
```

3. Justificación técnica de la solución aplicada

Protege los archivos físicos de la base en caso de robo o copia.

TDE cifra de manera transparente sin cambiar aplicaciones ni consultas existentes.

4. Explicación de las buenas prácticas utilizadas en el proyecto

Cifrado de datos en reposo.

Uso de master key y certificado único para cada base.

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

1. Enunciado del ejercicio

Configurar un ejemplo de Always Encrypted para la columna Precio Proveedor (o crear una nueva columna Precio Proveedor_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

2. Script de la solución en T-SQL

```
- Crear Column Master Key
CREATE COLUMN MASTER KEY
CMK_Qhatu
WITH
(
    KEY_STORE_PROVIDER_NAME
=
'MSSQL_CERTIFICATE_STORE',
    KEY_PATH =
'CurrentUser/My/Certificado
_CM_Key'
);
GO
-- Crear Column Encryption Key
CREATE COLUMN ENCRYPTION KEY CEK_Qhatu
WITH VALUES
(
    COLUMN_MASTER_KEY = CMK_Qhatu,
    ALGORITHM = 'RSA_OAEP',
    ENCRYPTED_VALUE = 0x... -- generado
automáticamente
);
GO
-- Crear columna cifrada
ALTER TABLE INVENTARIO
ADD PrecioProveedor_ENC decimal(10,2)
    COLLATE Latin1_General_BIN2
    ENCRYPTED WITH
    (COLUMN_ENCRYPTION_KEY = CEK_Qhatu,
ENCRYPTION_TYPE = DETERMINISTIC);
GO
```

3. Justificación técnica de la solución aplicada

Protege datos sensibles en reposo y en memoria, manteniendo confidencialidad incluso si la base es copiada.

Solo aplicaciones con acceso a CEK pueden leer los datos.

4. Explicación de las buenas prácticas utilizadas en el proyecto

Cifrado a nivel de columna para protección mínima necesaria.

Uso de CMK almacenada en certificado seguro.

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).

2. Script de la solución en T-SQL

```
-- Crear Server Audit
CREATE SERVER AUDIT Audit_Login
TO FILE (FILEPATH =
'C:\SQLAudits\', MAXSIZE = 10
MB)
WITH (ON_FAILURE = CONTINUE);
ALTER SERVER AUDIT Audit_Login
WITH (STATE = ON);
GO
```

```
-- Crear Database Audit
Specification
USE QhatuPeru;
GO
CREATE DATABASE AUDIT
SPECIFICATION Audit_DDL
FOR SERVER AUDIT Audit_Login
ADD
(SCHEMA_OBJECT_CHANGE_GROUP),
ADD
(DATABASE_OBJECT_PERMISSION_CHA
NGE_GROUP)
WITH (STATE = ON);
GO
```

3. Justificación técnica de la solución aplicada

Monitorea accesos y cambios críticos, proporcionando trazabilidad y detección de incidentes.

Diferencia entre login al servidor y cambios de objetos dentro de la base.

4. Explicación de las buenas prácticas utilizadas en el proyecto

Auditoría continua y centralizada.

Separación entre logs de servidor y base de datos.

Evita modificaciones no autorizadas y facilita auditoría de seguridad.