Proyecto 1: Creación y distribución de archivos físicos de la base QhatuPeru

1. Enunciado del ejercicio

Crear la base de datos QhatuPeru con un archivo primario, un archivo secundario y el log de transacciones en rutas distintas, asegurando mejor rendimiento y seguridad en el almacenamiento.

Código en T-SQL

```
CREATE DATABASE QhatuPeru
ON PRIMARY
(
    NAME = 'QhatuPeru_Primary',
    FILENAME = 'C:\SQLData\QhatuPeru_Primary.mdf',
    SIZE = 50MB,
    MAXSIZE = 500MB,
    FILEGROWTH = 10MB
),
FILEGROUP FG_Secundario
(
    NAME = 'QhatuPeru_Secundario',
    FILENAME = 'C:\SQLData\QhatuPeru_Secundario.ndf',
    SIZE = 30MB,
    MAXSIZE = 300MB,
    FILEGROWTH = 5MB
)
LOG ON
(
    NAME = 'QhatuPeru_Log',
    FILENAME = 'C:\SQLLogs\QhatuPeru_Log.ldf',
    SIZE = 25MB,
    MAXSIZE = 250MB,
    FILEGROWTH = 5MB
);
GO

-- Verificar creación
SELECT name, physical_name AS FilePath, type_desc
FROM sys.master_files
WHERE database_id = DB_ID('QhatuPeru');
GO
```

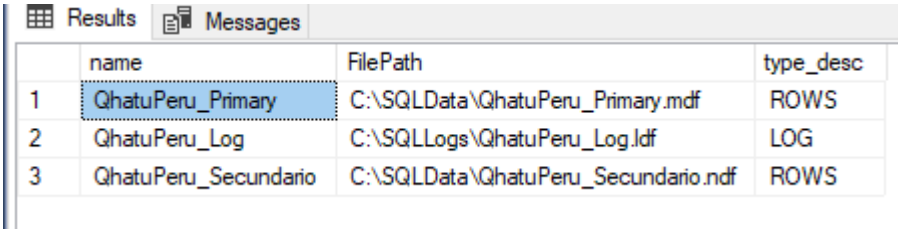
3. Justificación técnica de la solución aplicada

La distribución de archivos en diferentes rutas físicas mejora el rendimiento porque permite distribuir las operaciones de I/O entre múltiples discos. El archivo primario contiene las tablas del sistema, el secundario puede alojar tablas de usuario grandes, y el log se mantiene separado para optimizar las

operaciones de escritura transaccional. La configuración de FILEGROWTH automático evita bloqueos por falta de espacio, mientras que MAXSIZE previene el crecimiento descontrolado.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- **Separación de archivos**: Los archivos de datos y log están en discos diferentes para evitar contención de I/O
- **Filegroups personalizados**: Permiten organizar objetos de base de datos según su uso
- **Crecimiento controlado**: FILEGROWTH y MAXSIZE evitan fragmentación y saturación de disco
- **Nomenclatura clara**: Nombres descriptivos facilitan la administración
- **Dimensionamiento inicial**: SIZE apropiado reduce crecimientos frecuentes



Results | Messages

	name	FilePath	type_desc
1	QhatuPeru_Primary	C:\SQLData\QhatuPeru_Primary.mdf	ROWS
2	QhatuPeru_Log	C:\SQLLogs\QhatuPeru_Log.ldf	LOG
3	QhatuPeru_Secundario	C:\SQLData\QhatuPeru_Secundario.ndf	ROWS

Proyecto 2: Ajuste de configuración y validación de propiedades de QhatuPeru

1. Enunciado del ejercicio

Modificar la colación de la base de datos para soportar tildes y configurar el crecimiento automático del archivo de datos, evitando saturaciones inesperadas.

Código en T-SQL

```
-- a) Consultar propiedades actuales, modificar colación y configurar
crecimiento automático
USE master;
GO

-- Verificar propiedades actuales
SELECT
    name AS BaseDatos,
    collation_name AS Colacion,
    recovery_model_desc AS ModeloRecuperacion
FROM sys.databases
WHERE name = 'QhatuPeru';
GO

-- Modificar la colación de la base de datos
ALTER DATABASE QhatuPeru
COLLATE Modern_Spanish_CI_AS;
GO
```

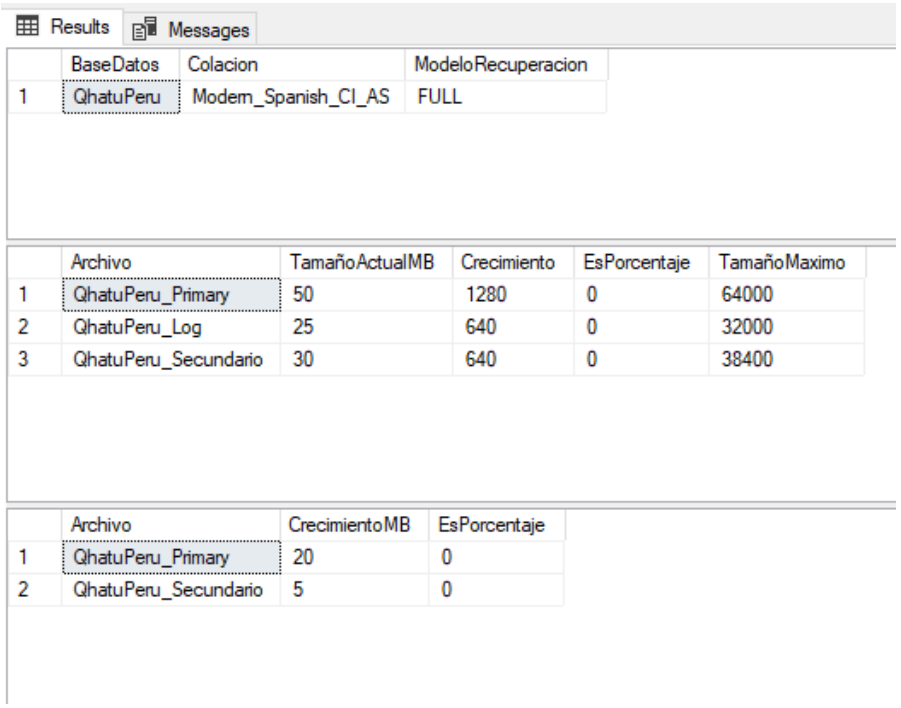
```
-- Verificar archivos actuales
USE QhatuPeru;
GO

SELECT
    name AS Archivo,
    size * 8 / 1024 AS TamañoActualMB,
    growth AS Crecimiento,
    is_percent_growth AS EsPorcentaje,
    max_size AS TamañoMaximo
FROM sys.database_files;
GO

-- b) Modificar el crecimiento automático del archivo primario a 20 MB
ALTER DATABASE QhatuPeru
MODIFY FILE
(
    NAME = 'QhatuPeru_Primary',
    FILEGROWTH = 20MB
);
GO

-- Verificar el cambio realizado
SELECT
    name AS Archivo,
    growth * 8 / 1024 AS CrecimientoMB,
    is_percent_growth AS EsPorcentaje
FROM sys.database_files
WHERE type_desc = 'ROWS';
GO
```

3. Justificación técnica de la solución aplicada

La colación Modern_Spanish_CI_AS permite el correcto almacenamiento y comparación de caracteres con tildes y eñes, esencial para una aplicación en español. Cambiar el crecimiento automático a 20 MB en lugar de porcentaje reduce la fragmentación del archivo y mejora el rendimiento, ya que crecimientos pequeños frecuentes pueden degradar el sistema.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- **Colación apropiada**: Modern_Spanish_CI_AS soporta caracteres latinos y es case-insensitive
- **Crecimiento en MB vs porcentaje**: Valores fijos son más predecibles y evitan crecimientos exponenciales
- **Validación pre y post cambio**: Consultas de verificación aseguran que los cambios se aplicaron correctamente
- **Documentación del estado actual**: Siempre verificar configuración antes de modificar



Results Messages

	BaseDatos	Colacion	ModeloRecuperacion
1	QhatuPeru	Modern_Spanish_CI_AS	FULL

	Archivo	TamañoActualMB	Crecimiento	EsPorcentaje	TamañoMaximo
1	QhatuPeru_Primary	50	1280	0	64000
2	QhatuPeru_Log	25	640	0	32000
3	QhatuPeru_Secundario	30	640	0	38400

	Archivo	CrecimientoMB	EsPorcentaje
1	QhatuPeru_Primary	20	0
2	QhatuPeru_Secundario	5	0

Proyecto 3: Definición de modelo de recuperación y respaldo para QhatuPeru

1. Enunciado del ejercicio

En el área de operaciones se identificaron diferentes necesidades de recuperación de datos según el proceso. Configurar el modelo de recuperación adecuado y ejecutar un respaldo de la base de datos.

código en T-SQL

```
-- a) Cambiar modelo de recuperación de QhatuPeru a Simple y luego a
Bulk-Logged
USE master;
GO

-- Verificar modelo actual
SELECT
    name AS BaseDatos,
    recovery_model_desc AS ModeloRecuperacion
FROM sys.databases
WHERE name = 'QhatuPeru';
GO

-- Cambiar a modelo SIMPLE
ALTER DATABASE QhatuPeru
SET RECOVERY SIMPLE;
GO

-- Verificar el cambio
SELECT
```

```
    name AS BaseDatos,
    recovery_model_desc AS ModeloRecuperacion
FROM sys.databases
WHERE name = 'QhatuPeru';
GO

-- Cambiar a modelo BULK_LOGGED
ALTER DATABASE QhatuPeru
SET RECOVERY BULK_LOGGED;
GO

-- Verificar el cambio final
SELECT
    name AS BaseDatos,
    recovery_model_desc AS ModeloRecuperacion
FROM sys.databases
WHERE name = 'QhatuPeru';
GO

-- Explicación de diferencias prácticas entre ambos modelos:
/*
SIMPLE:
- El log se trunca automáticamente
- No permite respaldos de log de transacciones
- Menor uso de espacio en disco
- Pérdida de datos desde el último respaldo completo/diferencial
- Ideal para entornos de desarrollo o datos no críticos

BULK_LOGGED:
- Registra mínimamente operaciones masivas (BULK INSERT, SELECT INTO)
- Permite respaldos de log de transacciones
- Mejor rendimiento en cargas masivas que FULL
- Pérdida mínima de datos con estrategia adecuada de respaldos
- Ideal para operaciones ETL y cargas masivas en producción
*/

-- b) Realizar un respaldo completo después de cambiar al modelo FULL
ALTER DATABASE QhatuPeru
SET RECOVERY FULL;
GO

-- Crear directorio para respaldos (ejecutar desde línea de comandos o
validar que existe)
-- EXEC xp_cmdshell 'mkdir C:\SQLBackups';

-- Respaldo completo de la base de datos
BACKUP DATABASE QhatuPeru
TO DISK = 'C:\SQLBackups\QhatuPeru_Full.bak'
WITH
    FORMAT,
    NAME = 'QhatuPeru-Respaldo Completo',
    DESCRIPTION = 'Respaldo completo después de configurar modelo FULL',
    COMPRESSION,
```

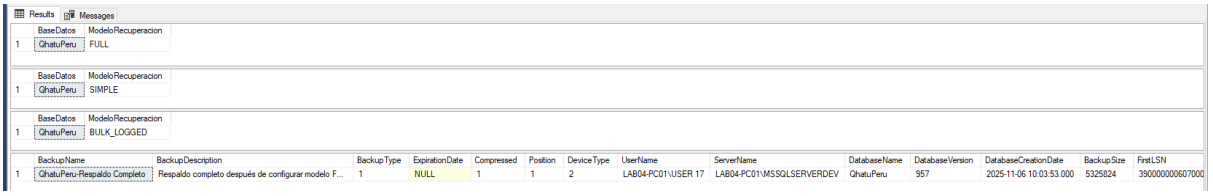
```
    STATS = 10;
GO

-- Verificar el respaldo realizado
RESTORE HEADERONLY
FROM DISK = 'C:\SQLBackups\QhatuPeru_Full.bak';
GO
```

3. Justificación técnica de la solución aplicada

El modelo de recuperación SIMPLE es útil para minimizar el tamaño del log pero no permite recuperación point-in-time. BULK_LOGGED optimiza operaciones masivas manteniendo capacidad de recuperación. FULL ofrece la máxima protección permitiendo restaurar hasta cualquier punto en el tiempo mediante respaldos de log. El respaldo completo es el punto de partida para cualquier estrategia de recuperación.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- **Verificación del estado actual**: Consultar antes de modificar
- **Documentación de diferencias**: Comentarios explican cada modelo
- **Respaldo con compresión**: Reduce espacio y tiempo de respaldo
- **Nomenclatura descriptiva**: Nombres claros para archivos de respaldo
- **Validación del respaldo**: RESTORE HEADERONLY confirma que el respaldo es válido
- **Uso de STATS**: Muestra progreso del respaldo


Results Messages

	BaseDatos	ModeloRecuperacion
1	QhatuPeru	FULL

	BaseDatos	ModeloRecuperacion
1	QhatuPeru	SIMPLE

	BaseDatos	ModeloRecuperacion
1	QhatuPeru	BULK_LOGGED

	BackupName	BackupDescription	BackupType	ExpirationDate	Compressed	Position	DeviceType	UserName	ServerName	DatabaseName	DatabaseVersion	DatabaseCreationDate	BackupSize	FirstLSN
1	QhatuPeru-Respaldo Completo	Respaldo completo después de configurar modelo F...	1	NULL	1	1	2	LAB04-PC01\USER 17	LAB04-PC01\MSSQLSERVERDEV	QhatuPeru	957	2025-11-06 10:03:53.000	5325824	39000000060700

Proyecto 4: Implementación de roles y usuarios para seguridad en QhatuPeru

1. Enunciado del ejercicio

Con el crecimiento de los equipos de ventas y atención al cliente, QhatuPeru solicita la creación de usuarios con roles diferenciados: cajeros acceden solo a consulta, administradores tienen control total y el gerente requiere acceso a reportes.

Código T-SQL

```
-- ================================================
-- CREACIÓN DE LOGINS, USUARIOS Y ASIGNACIÓN DE ROLES
-- ================================================

USE master;
GO

-- Crear login para Vendedor/Cajero
```

```
IF NOT EXISTS (SELECT * FROM sys.server_principals WHERE name =
'VendedorQhatu')
BEGIN
    CREATE LOGIN VendedorQhatu
    WITH PASSWORD = 'Vendedor2024!',
         DEFAULT_DATABASE = QhatuPeru,
         CHECK_POLICY = ON,
         CHECK_EXPIRATION = ON;
END
GO

-- Crear login para Consultor/Cliente
IF NOT EXISTS (SELECT * FROM sys.server_principals WHERE name =
'ConsultorCliente')
BEGIN
    CREATE LOGIN ConsultorCliente
    WITH PASSWORD = 'Consultor2024!',
         DEFAULT_DATABASE = QhatuPeru,
         CHECK_POLICY = ON,
         CHECK_EXPIRATION = ON;
END
GO

-- Cambiar contexto a la base de datos QhatuPeru
USE QhatuPeru;
GO

-- Crear usuarios asociados a los logins
IF NOT EXISTS (SELECT * FROM sys.database_principals WHERE name =
'VendedorQhatu')
    CREATE USER VendedorQhatu FOR LOGIN VendedorQhatu;
GO

IF NOT EXISTS (SELECT * FROM sys.database_principals WHERE name =
'ConsultorCliente')
    CREATE USER ConsultorCliente FOR LOGIN ConsultorCliente;
GO

-- ================================================
-- Asignación de roles y permisos
-- ================================================

-- a) Vendedor: puede leer y escribir, pero no eliminar
ALTER ROLE db_datawriter ADD MEMBER VendedorQhatu;
ALTER ROLE db_datareader ADD MEMBER VendedorQhatu;

GRANT INSERT, UPDATE, SELECT ON SCHEMA::dbo TO VendedorQhatu;
DENY DELETE ON SCHEMA::dbo TO VendedorQhatu;
GO

-- b) Consultor: solo lectura
ALTER ROLE db_datareader ADD MEMBER ConsultorCliente;
DENY INSERT, UPDATE, DELETE ON SCHEMA::dbo TO ConsultorCliente;
```

```
GO

-- ================================================
-- Verificación de usuarios y roles
-- ================================================
SELECT
    dp.name AS Usuario,
    dp.type_desc AS TipoUsuario,
    r.name AS Rol
FROM sys.database_principals dp
LEFT JOIN sys.database_role_members drm ON dp.principal_id =
drm.member_principal_id
LEFT JOIN sys.database_principals r ON drm.role_principal_id = r.principal_id
WHERE dp.type IN ('S', 'U')
  AND dp.name IN ('VendedorQhatu', 'ConsultorCliente');
GO

-- Verificar permisos explícitos otorgados o denegados
SELECT
    USER_NAME(grantee_principal_id) AS Usuario,
    permission_name AS Permiso,
    state_desc AS Estado,
    OBJECT_NAME(major_id) AS Objeto
FROM sys.database_permissions
WHERE grantee_principal_id IN (
    USER_ID('VendedorQhatu'),
    USER_ID('ConsultorCliente')
);
GO
```

3. Justificación técnica de la solución aplicada

La implementación de seguridad basada en roles sigue el principio de mínimo privilegio. VendedorQhatu puede leer y escribir datos necesarios para ventas pero no puede eliminar registros, protegiendo la integridad histórica. ConsultorCliente tiene acceso de solo lectura, apropiado para consultas sin riesgo de modificación accidental. Los logins con CHECK_POLICY aseguran contraseñas robustas.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- **Principio de mínimo privilegio**: Cada usuario tiene solo los permisos necesarios
- **Separación de roles**: Distinción clara entre lectura y escritura
- **Políticas de contraseña**: CHECK_POLICY y CHECK_EXPIRATION mejoran seguridad
- **DENY explícito**: Previene escalación accidental de privilegios
- **Verificación de permisos**: Consultas para auditar configuración de seguridad
- **Roles de base de datos**: Facilitan administración y mantenimiento

Results | Messages

	Usuario	TipoUsuario	Rol
1	VendedorQhatu	SQL_USER	db_datareader
2	VendedorQhatu	SQL_USER	db_datawriter
3	ConsultorCliente	SQL_USER	db_datareader

	Usuario	Permiso	Estado	Objeto
1	VendedorQhatu	CONNECT	GRANT	NULL
2	ConsultorCliente	CONNECT	GRANT	NULL
3	VendedorQhatu	DELETE	DENY	NULL
4	VendedorQhatu	INSERT	GRANT	NULL
5	VendedorQhatu	SELECT	GRANT	NULL
6	VendedorQhatu	UPDATE	GRANT	NULL
7	ConsultorCliente	DELETE	DENY	NULL
8	ConsultorCliente	INSERT	DENY	NULL
9	ConsultorCliente	UPDATE	DENY	NULL

Proyecto 5: Configuración granular de permisos en el módulo de ventas de QhatuPeru

1. Enunciado del ejercicio

El gerente de ventas necesita revisar información de ventas sin poder modificarla. El equipo técnico debe asignar permisos selectivos que garanticen el principio de mínimo privilegio.

Código T-SQL

```
USE QhatuPeru;
GO

-- Crear login y usuario para el Gerente
USE master;
GO

CREATE LOGIN GerenteQhatu
WITH PASSWORD = 'Gerente2024!',
     DEFAULT_DATABASE = QhatuPeru,
     CHECK_POLICY = ON,
     CHECK_EXPIRATION = ON;
GO

USE QhatuPeru;
GO

CREATE USER GerenteQhatu FOR LOGIN GerenteQhatu;
GO

-- Crear login y usuario para Cajero
```

```
USE master;
GO

CREATE LOGIN CajeroQhatu
WITH PASSWORD = 'Cajero2024!',
     DEFAULT_DATABASE = QhatuPeru,
     CHECK_POLICY = ON,
     CHECK_EXPIRATION = ON;
GO

USE QhatuPeru;
GO

CREATE USER CajeroQhatu FOR LOGIN CajeroQhatu;
GO

-- Crear tablas necesarias para el ejercicio
CREATE TABLE Reportes (
    ReporteID INT PRIMARY KEY IDENTITY(1,1),
    NombreReporte VARCHAR(100),
    FechaGeneracion DATETIME DEFAULT GETDATE(),
    Contenido VARCHAR(MAX)
);
GO

CREATE TABLE Ventas (
    VentaID INT PRIMARY KEY IDENTITY(1,1),
    FechaVenta DATETIME DEFAULT GETDATE(),
    ClienteID INT,
    MontoTotal DECIMAL(10,2),
    Estado VARCHAR(20)
);
GO

-- a) Otorgar a GerenteQhatu acceso exclusivo (solo SELECT) a la tabla
Reportes
GRANT SELECT ON dbo.Reportes TO GerenteQhatu;
GO

-- Denegar cualquier modificación
DENY INSERT, UPDATE, DELETE ON dbo.Reportes TO GerenteQhatu;
GO

-- b) Revocar a CajeroQhatu el permiso UPDATE sobre la tabla Ventas
-- Primero otorgar algunos permisos básicos al cajero
GRANT SELECT, INSERT ON dbo.Ventas TO CajeroQhatu;
GO

-- Revocar específicamente UPDATE
REVOKE UPDATE ON dbo.Ventas TO CajeroQhatu;
GO

-- Verificar permisos de GerenteQhatu
```

```
SELECT
    'GerenteQhatu' AS Usuario,
    OBJECT_NAME(major_id) AS Tabla,
    permission_name AS Permiso,
    state_desc AS Estado
FROM sys.database_permissions
WHERE grantee_principal_id = USER_ID('GerenteQhatu')
      AND major_id = OBJECT_ID('Reportes');
GO

-- Verificar permisos de CajeroQhatu
SELECT
    'CajeroQhatu' AS Usuario,
    OBJECT_NAME(major_id) AS Tabla,
    permission_name AS Permiso,
    state_desc AS Estado
FROM sys.database_permissions
WHERE grantee_principal_id = USER_ID('CajeroQhatu')
      AND major_id = OBJECT_ID('Ventas');
GO

-- Prueba de permisos (ejecutar como cada usuario para validar)
-- Como GerenteQhatu:
EXECUTE AS USER = 'GerenteQhatu';
SELECT * FROM Reportes; -- Debe funcionar
-- INSERT INTO Reportes VALUES ('Test'); -- Debe fallar
REVERT;
GO

-- Como CajeroQhatu:
EXECUTE AS USER = 'CajeroQhatu';
SELECT * FROM Ventas; -- Debe funcionar
INSERT INTO Ventas (ClienteID, MontoTotal, Estado) VALUES (1, 100.00,
'Completada'); -- Debe funcionar
-- UPDATE Ventas SET MontoTotal = 200 WHERE VentaID = 1; -- Debe fallar
REVERT;
GO
```

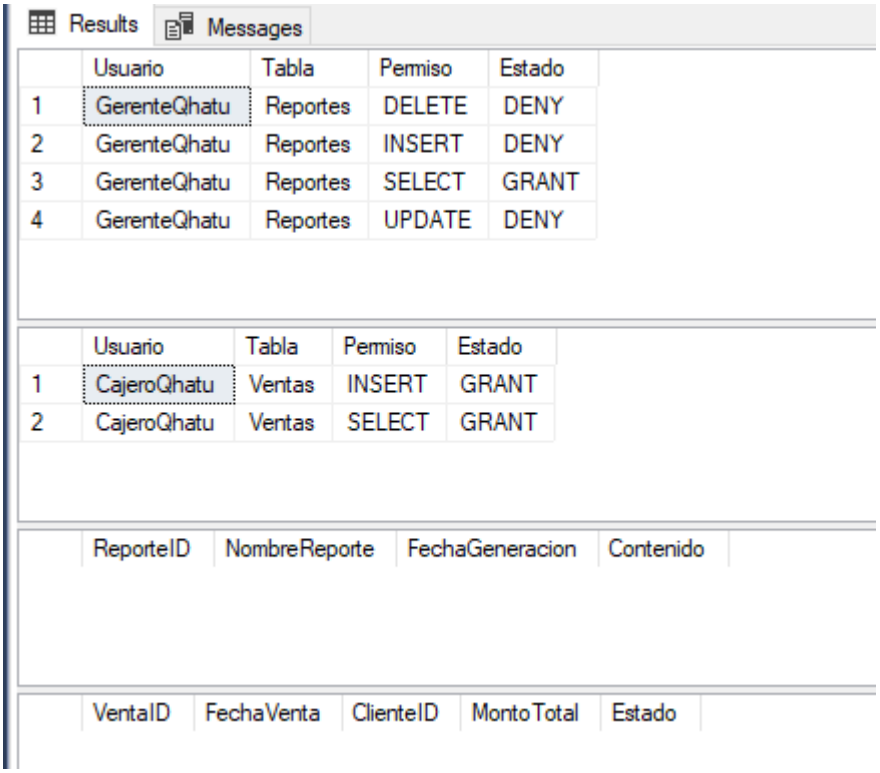
3. Justificación técnica de la solución aplicada

La configuración granular de permisos a nivel de tabla permite control preciso sobre las operaciones que cada usuario puede realizar. GRANT SELECT para el gerente garantiza acceso de solo lectura a reportes, mientras que DENY previene cualquier modificación accidental. REVOKE UPDATE en Ventas para el cajero evita alteración de transacciones completadas, manteniendo integridad de datos históricos.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- **Permisos a nivel de objeto**: Control granular sobre tablas específicas
- **DENY vs REVOKE**: DENY tiene precedencia y bloquea explícitamente acciones
- **Pruebas de permisos**: EXECUTE AS valida la configuración sin cambiar de sesión
- **Documentación de permisos**: Consultas de verificación documentan el estado
- **Separación de funciones**: Cada rol tiene acceso apropiado a su función

- **Auditoría de permisos**: sys.database_permissions permite revisiones periódicas



Results Messages

	Usuario	Tabla	Permiso	Estado
1	GerenteQhatu	Reportes	DELETE	DENY
2	GerenteQhatu	Reportes	INSERT	DENY
3	GerenteQhatu	Reportes	SELECT	GRANT
4	GerenteQhatu	Reportes	UPDATE	DENY

	Usuario	Tabla	Permiso	Estado
1	CajeroQhatu	Ventas	INSERT	GRANT
2	CajeroQhatu	Ventas	SELECT	GRANT

	ReporteID	NombreReporte	FechaGeneracion	Contenido

	VentaID	FechaVenta	ClienteID	MontoTotal	Estado

Proyecto 6: Identificación y solución de procesos lentos en QhatuPeru

1. Enunciado del ejercicio

Durante una campaña promocional, el sistema presenta lentitud. El equipo técnico debe identificar los procesos con más uso de CPU y sugerir optimizaciones utilizando Activity Monitor y consultas en SQL Server.

Código T-SQL

```
-- a) Identificar los 3 procesos con mayor consumo de CPU
SELECT TOP 3
    r.session_id AS SessionID,
    s.login_name AS Usuario,
    s.host_name AS Host,
    DB_NAME(r.database_id) AS BaseDatos,
    r.cpu_time AS TiempoCPU_ms,
    r.total_elapsed_time AS TiempoTotal_ms,
    r.reads AS Lecturas,
    r.writes AS Escrituras,
    r.logical_reads AS LecturasLogicas,
    r.status AS Estado,
    r.command AS Comando,
    r.wait_type AS TipoEspera,
    r.wait_time AS TiempoEspera_ms,
    r.last_wait_type AS UltimaEspera,
    SUBSTRING(t.TEXT, 1, 4000) AS ConsultaSQL
```

```
FROM sys.dm_exec_requests r
INNER JOIN sys.dm_exec_sessions s
    ON r.session_id = s.session_id
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE r.session_id > 50  -- Excluir procesos del sistema
ORDER BY r.cpu_time DESC;
GO

-- ================================================
-- b) Consultar bloqueos activos en la base de datos
-- ================================================
SELECT
    blocking.session_id AS SesionBloqueante,
    blocked.session_id AS SesionBloqueada,
    s1.login_name AS UsuarioBloqueante,
    s2.login_name AS UsuarioBloqueado,
    DB_NAME(blocking.database_id) AS BaseDatos,
    blocking.wait_type AS TipoEspera,
    blocking.wait_time AS TiempoEspera_ms,
    blocking.wait_resource AS RecursoBloqueado,
    SUBSTRING(bt.TEXT, 1, 4000) AS ConsultaBloqueante,
    SUBSTRING(blt.TEXT, 1, 4000) AS ConsultaBloqueada
FROM sys.dm_exec_requests blocked
INNER JOIN sys.dm_exec_requests blocking
    ON blocked.blocking_session_id = blocking.session_id
INNER JOIN sys.dm_exec_sessions s1
    ON blocking.session_id = s1.session_id
INNER JOIN sys.dm_exec_sessions s2
    ON blocked.session_id = s2.session_id
CROSS APPLY sys.dm_exec_sql_text(blocking.sql_handle) bt
CROSS APPLY sys.dm_exec_sql_text(blocked.sql_handle) blt
WHERE blocked.blocking_session_id <> 0;
GO

-- ================================================
-- c) Consulta alternativa de bloqueos simples
-- ================================================
SELECT
    t1.resource_type AS TipoRecurso,
    DB_NAME(resource_database_id) AS BaseDatos,
    t1.resource_associated_entity_id AS RecursoID,
    t1.request_mode AS ModoSolicitud,
    t1.request_session_id AS SesionID,
    t2.blocking_session_id AS SesionBloqueante,
    t2.wait_type AS TipoEspera,
    t2.wait_duration_ms AS DuracionEspera_ms
FROM sys.dm_tran_locks t1
INNER JOIN sys.dm_os_waiting_tasks t2
    ON t1.lock_owner_address = t2.resource_address
WHERE t2.blocking_session_id IS NOT NULL;
GO

-- ================================================
```

```
-- d) Información detallada de sesiones activas
-- ============================================
SELECT
    s.session_id AS SessionID,
    s.login_name AS Usuario,
    s.host_name AS Host,
    s.program_name AS Programa,
    DB_NAME(r.database_id) AS BaseDatos,
    r.status AS Estado,
    r.command AS Comando,
    r.cpu_time AS TiempoCPU,
    r.total_elapsed_time AS TiempoTotal,
    r.blocking_session_id AS BloqueadaPor,
    SUBSTRING(t.TEXT, 1, 4000) AS ConsultaActual
FROM sys.dm_exec_sessions s
LEFT JOIN sys.dm_exec_requests r
    ON s.session_id = r.session_id
OUTER APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE s.is_user_process = 1
ORDER BY r.cpu_time DESC;
GO

-- ============================================
-- e) Estadísticas de espera del sistema
-- ============================================
SELECT TOP 10
    wait_type AS TipoEspera,
    wait_time_ms / 1000.0 AS TiempoEspera_seg,
    waiting_tasks_count AS TareasEnEspera,
    wait_time_ms / waiting_tasks_count AS PromedioEspera_ms
FROM sys.dm_os_wait_stats
WHERE wait_type NOT LIKE '%SLEEP%'
  AND waiting_tasks_count > 0
ORDER BY wait_time_ms DESC;
GO
```

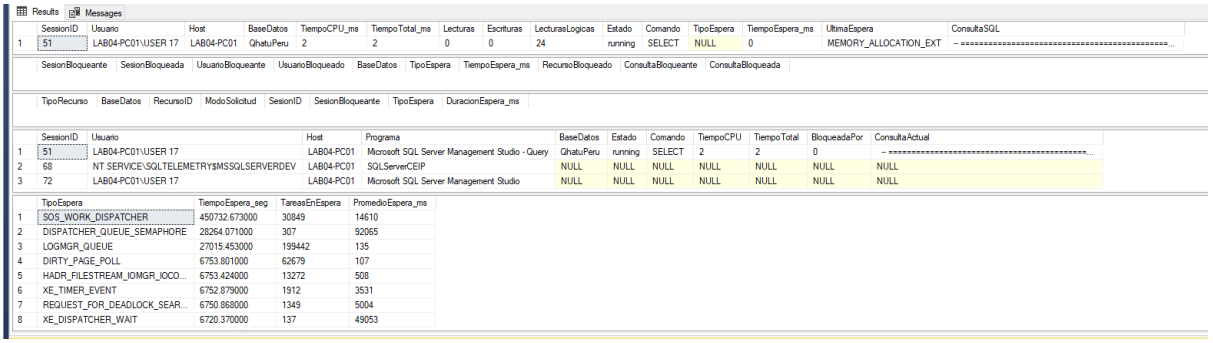
3. Justificación técnica de la solución aplicada

Activity Monitor proporciona una vista en tiempo real del rendimiento del servidor, facilitando la identificación rápida de procesos problemáticos. Las vistas de administración dinámica (DMVs) como sys.dm_exec_requests y sys.dm_tran_locks permiten análisis programático de procesos y bloqueos. Identificar consumo de CPU, lecturas lógicas y esperas ayuda a priorizar optimizaciones: índices faltantes, consultas ineficientes o contención de recursos.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- Múltiples métodos de diagnóstico: Activity Monitor (GUI) y DMVs (scripts)
- Filtrado de procesos del sistema: session_id > 50 excluye procesos internos
- Análisis de métricas clave: CPU, I/O, tiempo de espera, bloqueos
- Identificación de consultas: sys.dm_exec_sql_text muestra SQL en ejecución
- Priorización TOP N: Enfocarse en los mayores consumidores de recursos

- Documentación de esperas: Tipos de espera indican cuello de botella específico



Results Messages

SesionID	Usuario	Host	BaseDatos	TiempoCPU_ms	TiempoTotal_ms	Lecturas	Escrituras	LecturasLogicas	Estado	Comando	TipoEspera	TiempoEspera_ms	UltimaEspera	ConsultaSQL
1 51	LAB04-PC01\USER 17	LAB04-PC01	QhatuPeru	2	2	0	0	24	running	SELECT	NULL	0	MEMORY_ALLOCATION_EXT	-- ==========

SesionBloqueante	SesionBloqueada	UsuarioBloqueante	UsuarioBloqueado	BaseDatos	TipoEspera	TiempoEspera_ms	RecursoBloqueado	ConsultaBloqueante	ConsultaBloqueada

TipoRecurso	BaseDatos	RecursoID	ModoSolicitud	SesionID	SesionBloqueante	TipoEspera	DuracionEspera_ms

SessionID	Usuario	Host	Programa	BaseDatos	Estado	Comando	TiempoCPU	TiempoTotal	BloqueadaPor	ConsultaActual
1 51	LAB04-PC01\USER 17	LAB04-PC01	Microsoft SQL Server Management Studio - Query	QhatuPeru	running	SELECT	2	2	0	-- ==========
2 68	NT SERVICE\SQLTELEMETRY$MSSQLSERVERDEV	LAB04-PC01	SQLServerCEIP	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3 72	LAB04-PC01\USER 17	LAB04-PC01	Microsoft SQL Server Management Studio	NULL	NULL	NULL	NULL	NULL	NULL	NULL

TipoEspera	TiempoEspera_seg	TareasEnEspera	PromedioEspera_ms
1 SOS_WORK_DISPATCHER	450732.673000	30849	14610
2 DISPATCHER_QUEUE_SEMAPHORE	28264.071000	307	92065
3 LOGMGR_QUEUE	27015.453000	199442	135
4 DIRTY_PAGE_POLL	6753.801000	62679	107
5 HADR_FILESTREAM_IOMGR_IOCO...	6753.424000	13272	508
6 XE_TIMER_EVENT	6752.879000	1912	3531
7 REQUEST_FOR_DEADLOCK_SEAR...	6750.868000	1349	5004
8 XE_DISPATCHER_WAIT	6720.370000	137	49053

Proyecto 7: Automatización de respaldos y limpieza del sistema QhatuPeru

1. Enunciado del ejercicio

Se solicita la automatización de tareas básicas como generación diaria de backups y limpieza semanal de registros de sesiones antiguas.

Código T-SQL

```
-- a) Crear un Job en SQL Server Agent que realice respaldo diario automático
de QhatuPeru
USE msdb;
GO

-- Verificar que SQL Server Agent esté habilitado
-- (esto debe hacerse desde SQL Server Configuration Manager)

-- Crear el Job de respaldo diario
EXEC dbo.sp_add_job
    @job_name = N'Respaldo_Diario_QhatuPeru',
    @enabled = 1,
    @description = N'Respaldo automático diario completo de la base de datos
QhatuPeru',
    @category_name = N'Database Maintenance';
GO

-- Agregar el paso del Job (el script de respaldo)
EXEC dbo.sp_add_jobstep
    @job_name = N'Respaldo_Diario_QhatuPeru',
    @step_name = N'Ejecutar_Respaldo_Completo',
    @subsystem = N'TSQL',
    @command = N'
DECLARE @BackupPath NVARCHAR(500);
DECLARE @FileName NVARCHAR(500);
DECLARE @Date NVARCHAR(50);
```

```
-- Generar nombre de archivo con fecha
SET @Date = CONVERT(NVARCHAR(50), GETDATE(), 112) + ''_'' +
REPLACE(CONVERT(NVARCHAR(50), GETDATE(), 108), '':'', '''');
SET @BackupPath = ''C:\SQLBackups\'';
SET @FileName = @BackupPath + ''QhatuPeru_Diario_'' + @Date + ''.bak'';

-- Realizar respaldo completo
BACKUP DATABASE QhatuPeru
TO DISK = @FileName
WITH
    FORMAT,
    NAME = ''QhatuPeru-Respaldo Diario Completo'',
    DESCRIPTION = ''Respaldo automático diario de QhatuPeru'',
    COMPRESSION,
    STATS = 10;

PRINT ''Respaldo completado: '' + @FileName;
',
    @database_name = N'QhatuPeru',
    @retry_attempts = 3,
    @retry_interval = 5;
GO

-- Configurar schedule (programación diaria a las 2:00 AM)
EXEC dbo.sp_add_schedule
    @schedule_name = N'Schedule_Diario_2AM',
    @enabled = 1,
    @freq_type = 4,  -- Diario
    @freq_interval = 1,  -- Cada 1 día
    @active_start_time = 020000;  -- 02:00:00 AM
GO

-- Asociar el schedule al Job
EXEC dbo.sp_attach_schedule
    @job_name = N'Respaldo_Diario_QhatuPeru',
    @schedule_name = N'Schedule_Diario_2AM';
GO

-- Agregar notificación al Job (servidor local)
EXEC dbo.sp_add_jobserver
    @job_name = N'Respaldo_Diario_QhatuPeru',
    @server_name = N'(LOCAL)';
GO

-- b) Diseñar y programar un Job que elimine registros de la tabla Sesiones
con más de 15 días de antigüedad cada semana
-- Primero crear la tabla Sesiones si no existe
USE QhatuPeru;
GO

IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Sesiones]'))
BEGIN
```

```
    CREATE TABLE Sesiones (
        SesionID INT PRIMARY KEY IDENTITY(1,1),
        UsuarioID INT NOT NULL,
        FechaInicio DATETIME NOT NULL DEFAULT GETDATE(),
        FechaFin DATETIME,
        IPAddress VARCHAR(50),
        EstadoSesion VARCHAR(20)
    );
END
GO

-- Crear el Job de limpieza semanal
USE msdb;
GO

EXEC dbo.sp_add_job
    @job_name = N'Limpieza_Semanal_Sesiones',
    @enabled = 1,
    @description = N'Eliminación semanal de registros de sesiones con más de
15 días de antigüedad',
    @category_name = N'Database Maintenance';
GO

-- Agregar el paso del Job
EXEC dbo.sp_add_jobstep
    @job_name = N'Limpieza_Semanal_Sesiones',
    @step_name = N'Eliminar_Sesiones_Antiguas',
    @subsystem = N'TSQL',
    @command = N'
USE QhatuPeru;
GO

DECLARE @FechaLimite DATETIME;
DECLARE @RegistrosEliminados INT;

-- Calcular fecha límite (15 días atrás)
SET @FechaLimite = DATEADD(DAY, -15, GETDATE());

-- Eliminar registros antiguos
DELETE FROM Sesiones
WHERE FechaInicio < @FechaLimite;

SET @RegistrosEliminados = @@ROWCOUNT;

-- Log del proceso
PRINT ''Limpieza completada: '' + CAST(@RegistrosEliminados AS VARCHAR(10)) +
'' registros eliminados'';
PRINT ''Fecha límite: '' + CONVERT(VARCHAR(50), @FechaLimite, 120);
',
    @database_name = N'QhatuPeru',
    @retry_attempts = 2,
    @retry_interval = 5;
GO
```

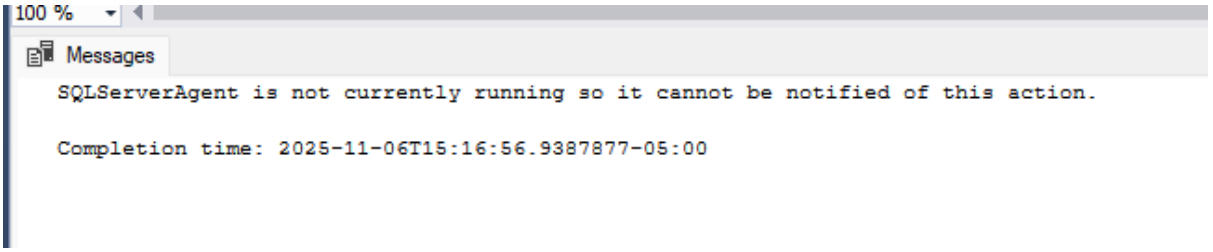
```
-- Configurar schedule (programación semanal los doming
```

3. Justificación técnica de la solución aplicada

SQL Server Agent Jobs permiten automatizar tareas críticas sin intervención manual, reduciendo errores humanos y garantizando consistencia. El respaldo diario a las 2:00 AM se ejecuta en horario de baja actividad, minimizando impacto en rendimiento. La nomenclatura con fecha/hora en archivos facilita identificación y rotación de respaldos. La limpieza semanal de sesiones antiguas optimiza el rendimiento de consultas y reduce el tamaño de la base de datos, manteniendo solo datos relevantes. El uso de DATEADD garantiza cálculo preciso de fechas independiente de zona horaria.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- **Automatización con SQL Server Agent**: Elimina dependencia de ejecución manual
- **Horarios de baja demanda**: Jobs programados en madrugada reducen contención
- **Nombres de archivo dinámicos**: Incluyen fecha/hora para evitar sobrescritura
- **COMPRESSION en respaldos**: Reduce espacio en disco y tiempo de respaldo
- **Reintentos configurados**: @retry_attempts permite recuperación de fallos temporales
- **Limpieza basada en antigüedad**: DATEADD(-15) mantiene datos recientes
- **Registro de actividad**: PRINT y @@ROWCOUNT documentan ejecución
- **Verificación de Jobs**: Consulta final valida configuración correcta



Proyecto 8: Registro de estados en pedidos en QhatuPeru

1. Enunciado del ejercicio

Con la finalidad de optimizar la trazabilidad de los envíos, se solicita agregar una columna "EstadoEnvio" en la tabla Pedidos, permitiendo identificar el estado actual de cada pedido.

Código T-SQL

USE QhatuPeru;
GO

```
-- Crear la tabla Pedidos si no existe
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Pedidos]'))
BEGIN
    CREATE TABLE Pedidos (
        PedidoID INT PRIMARY KEY IDENTITY(1,1),
```

```
        ClienteID INT NOT NULL,
        FechaPedido DATETIME DEFAULT GETDATE(),
        MontoTotal DECIMAL(10,2),
        DireccionEnvio VARCHAR(200),
        FechaCreacion DATETIME DEFAULT GETDATE()
    );

    -- Insertar datos de ejemplo
    INSERT INTO Pedidos (ClienteID, FechaPedido, MontoTotal, DireccionEnvio)
    VALUES
        (1, GETDATE(), 150.50, 'Av. Principal 123'),
        (2, GETDATE()-1, 280.00, 'Jr. Los Olivos 456'),
        (3, GETDATE()-2, 95.75, 'Calle Lima 789');
END
GO

-- a) Agregar la columna "Prioridad" tipo INT a la tabla Pedidos
ALTER TABLE Pedidos
ADD Prioridad INT NULL;
GO

-- Actualizar valores de prioridad para registros existentes
UPDATE Pedidos
SET Prioridad = CASE
    WHEN MontoTotal >= 200 THEN 1  -- Alta prioridad
    WHEN MontoTotal >= 100 THEN 2  -- Media prioridad
    ELSE 3  -- Baja prioridad
END;
GO

-- Verificar la columna agregada
SELECT
    PedidoID,
    ClienteID,
    MontoTotal,
    Prioridad,
    DireccionEnvio
FROM Pedidos;
GO

-- b) Eliminar la columna "Prioridad" de la tabla Pedidos
ALTER TABLE Pedidos
DROP COLUMN Prioridad;
GO

-- Verificar que la columna fue eliminada
SELECT
    COLUMN_NAME AS Columna,
    DATA_TYPE AS TipoDato,
    IS_NULLABLE AS Nullable
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Pedidos'
ORDER BY ORDINAL_POSITION;
```

```
GO

-- Agregar la columna EstadoEnvio como solución final
ALTER TABLE Pedidos
ADD EstadoEnvio VARCHAR(50) NULL
    CONSTRAINT DF_Pedidos_EstadoEnvio DEFAULT 'Pendiente';
GO

-- Agregar constraint para validar valores permitidos
ALTER TABLE Pedidos
ADD CONSTRAINT CK_Pedidos_EstadoEnvio
CHECK (EstadoEnvio IN ('Pendiente', 'En Proceso', 'Enviado', 'En Tránsito',
'Entregado', 'Cancelado'));
GO

-- Actualizar estados según antigüedad del pedido
UPDATE Pedidos
SET EstadoEnvio = CASE
    WHEN DATEDIFF(DAY, FechaPedido, GETDATE()) = 0 THEN 'Pendiente'
    WHEN DATEDIFF(DAY, FechaPedido, GETDATE()) = 1 THEN 'En Proceso'
    WHEN DATEDIFF(DAY, FechaPedido, GETDATE()) >= 2 THEN 'Enviado'
END;
GO

-- Consultar pedidos con su estado de envío
SELECT
    PedidoID,
    ClienteID,
    FechaPedido,
    MontoTotal,
    EstadoEnvio,
    DATEDIFF(DAY, FechaPedido, GETDATE()) AS DiasDesdeCreacion
FROM Pedidos
ORDER BY FechaPedido DESC;
GO

-- Crear índice para optimizar consultas por estado
CREATE NONCLUSTERED INDEX IX_Pedidos_EstadoEnvio
ON Pedidos(EstadoEnvio)
INCLUDE (PedidoID, FechaPedido, ClienteID);
GO
```

3. Justificación técnica de la solución aplicada

Agregar la columna EstadoEnvio con un valor DEFAULT 'Pendiente' asegura que todos los nuevos pedidos tengan un estado inicial automáticamente. El constraint CHECK garantiza integridad referencial limitando valores a estados válidos predefinidos, previniendo datos inconsistentes. VARCHAR(50) proporciona flexibilidad para descripciones legibles. El índice sobre EstadoEnvio optimiza consultas frecuentes de filtrado por estado, común en sistemas de seguimiento de pedidos.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- DEFAULT constraint: Automatiza asignación de estado inicial
- CHECK constraint: Valida valores permitidos a nivel de base de datos
- Tipo de dato apropiado: VARCHAR para valores descriptivos legibles
- Índice estratégico: IX_Pedidos_EstadoEnvio optimiza consultas de filtrado
- INCLUDE en índice: Columnas adicionales evitan lookups
- Validación de estructura: INFORMATION_SCHEMA documenta cambios
- Actualización inteligente: CASE/DATEDIFF asigna estados lógicos
- Nombres descriptivos: EstadoEnvio es autoexplicativo

Proyecto 9: Implementación de registros automáticos de modificaciones en QhatuPeru

1. Enunciado del ejercicio

Es necesario auditar todas las modificaciones realizadas en la tabla Clientes para cumplir con políticas internas de seguridad y protección de datos.

Código T-SQL

```
USE QhatuPeru;
GO

-- Crear la tabla Clientes si no existe
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Clientes]'))
BEGIN
    CREATE TABLE Clientes (
        ClienteID INT PRIMARY KEY IDENTITY(1,1),
        NombreCompleto VARCHAR(100) NOT NULL,
        Email VARCHAR(100),
        Telefono VARCHAR(20),
        Direccion VARCHAR(200),
        FechaRegistro DATETIME DEFAULT GETDATE(),
        Activo BIT DEFAULT 1
    );

    -- Insertar datos de ejemplo
    INSERT INTO Clientes (NombreCompleto, Email, Telefono, Direccion)
    VALUES
        ('Juan Pérez García', 'juan.perez@email.com', '987654321', 'Av.
Arequipa 1234'),
        ('María López Torres', 'maria.lopez@email.com', '956781234', 'Jr.
Cusco 567'),
        ('Carlos Mendoza Silva', 'carlos.mendoza@email.com', '912345678',
'Calle Lima 890');
END
GO

-- a) Crear tabla AuditoriaClientes para registrar cambios en Clientes
```

```
CREATE TABLE AuditoriaClientes (
    AuditoriaID INT PRIMARY KEY IDENTITY(1,1),
    ClienteID INT NOT NULL,
    Operacion VARCHAR(10) NOT NULL,  -- INSERT, UPDATE, DELETE
    NombreCompleto_Anterior VARCHAR(100),
    NombreCompleto_Nuevo VARCHAR(100),
    Email_Anterior VARCHAR(100),
    Email_Nuevo VARCHAR(100),
    Telefono_Anterior VARCHAR(20),
    Telefono_Nuevo VARCHAR(20),
    Direccion_Anterior VARCHAR(200),
    Direccion_Nuevo VARCHAR(200),
    Usuario VARCHAR(100) DEFAULT SYSTEM_USER,
    FechaModificacion DATETIME DEFAULT GETDATE(),
    HostName VARCHAR(100) DEFAULT HOST_NAME(),
    NombreAplicacion VARCHAR(100) DEFAULT APP_NAME()
);
GO

-- Crear índice para consultas de auditoría
CREATE NONCLUSTERED INDEX IX_AuditoriaClientes_ClienteID_Fecha
ON AuditoriaClientes(ClienteID, FechaModificacion DESC);
GO

-- b) Desarrollar trigger que registre en AuditoriaClientes cada eliminación
de registros en Clientes
CREATE TRIGGER TR_Clientes_Delete_Auditoria
ON Clientes
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO AuditoriaClientes (
        ClienteID,
        Operacion,
        NombreCompleto_Anterior,
        Email_Anterior,
        Telefono_Anterior,
        Direccion_Anterior
    )
    SELECT
        d.ClienteID,
        'DELETE',
        d.NombreCompleto,
        d.Email,
        d.Telefono,
        d.Direccion
    FROM deleted d;

    -- Log adicional
    PRINT 'Auditoría: ' + CAST(@@ROWCOUNT AS VARCHAR(10)) + ' eliminaciones
registradas en AuditoriaClientes';
```

```
END
GO

-- Crear trigger para UPDATE
CREATE TRIGGER TR_Clientes_Update_Auditoria
ON Clientes
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO AuditoriaClientes (
        ClienteID,
        Operacion,
        NombreCompleto_Anterior,
        NombreCompleto_Nuevo,
        Email_Anterior,
        Email_Nuevo,
        Telefono_Anterior,
        Telefono_Nuevo,
        Direccion_Anterior,
        Direccion_Nuevo
    )
    SELECT
        d.ClienteID,
        'UPDATE',
        d.NombreCompleto,
        i.NombreCompleto,
        d.Email,
        i.Email,
        d.Telefono,
        i.Telefono,
        d.Direccion,
        i.Direccion
    FROM deleted d
    INNER JOIN inserted i ON d.ClienteID = i.ClienteID;

    PRINT 'Auditoría: ' + CAST(@@ROWCOUNT AS VARCHAR(10)) + ' actualizaciones
registradas en AuditoriaClientes';
END
GO

-- Crear trigger para INSERT
CREATE TRIGGER TR_Clientes_Insert_Auditoria
ON Clientes
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO AuditoriaClientes (
        ClienteID,
        Operacion,
```

```
        NombreCompleto_Nuevo,
        Email_Nuevo,
        Telefono_Nuevo,
        Direccion_Nuevo
    )
    SELECT
        i.ClienteID,
        'INSERT',
        i.NombreCompleto,
        i.Email,
        i.Telefono,
        i.Direccion
    FROM inserted i;

    PRINT 'Auditoría: ' + CAST(@@ROWCOUNT AS VARCHAR(10)) + ' inserciones
registradas en AuditoriaClientes';
END
GO

-- Probar los triggers
-- Prueba INSERT
INSERT INTO Clientes (NombreCompleto, Email, Telefono, Direccion)
VALUES ('Ana Fernández Ruiz', 'ana.fernandez@email.com', '998877665', 'Av.
Grau 345');
GO

-- Prueba UPDATE
UPDATE Clientes
SET Email = 'juan.perez.nuevo@email.com',
    Telefono = '999888777'
WHERE ClienteID = 1;
GO

-- Prueba DELETE
DELETE FROM Clientes
WHERE ClienteID = 2;
GO

-- Consultar auditoría registrada
SELECT
    AuditoriaID,
    ClienteID,
    Operacion,
    CASE
        WHEN Operacion = 'INSERT' THEN NombreCompleto_Nuevo
        WHEN Operacion = 'DELETE' THEN NombreCompleto_Anterior
        ELSE NombreCompleto_Anterior + ' → ' + NombreCompleto_Nuevo
    END AS Cliente,
    Usuario,
    FechaModificacion,
    HostName,
    NombreAplicacion
FROM AuditoriaClientes
```

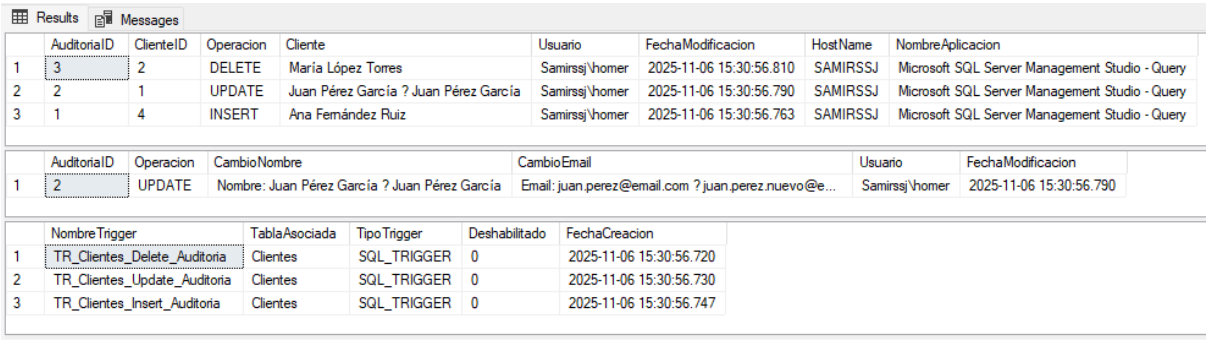
```
ORDER BY FechaModificacion DESC;
GO

-- Consultar cambios detallados de un cliente específico
SELECT
    AuditoriaID,
    Operacion,
    CASE
        WHEN NombreCompleto_Anterior IS NOT NULL AND NombreCompleto_Nuevo IS
NOT NULL
        THEN 'Nombre: ' + NombreCompleto_Anterior + ' → ' +
NombreCompleto_Nuevo
        WHEN NombreCompleto_Anterior IS NOT NULL
        THEN 'Eliminado: ' + NombreCompleto_Anterior
        ELSE 'Creado: ' + NombreCompleto_Nuevo
    END AS CambioNombre,
    CASE
        WHEN Email_Anterior IS NOT NULL AND Email_Nuevo IS NOT NULL
        THEN 'Email: ' + Email_Anterior + ' → ' + Email_Nuevo
        WHEN Email_Anterior IS NOT NULL
        THEN 'Email eliminado: ' + Email_Anterior
        WHEN Email_Nuevo IS NOT NULL
        THEN 'Email creado: ' + Email_Nuevo
    END AS CambioEmail,
    Usuario,
    FechaModificacion
FROM AuditoriaClientes
WHERE ClienteID = 1
ORDER BY FechaModificacion DESC;
GO

-- Verificar triggers creados
SELECT
    name AS NombreTrigger,
    OBJECT_NAME(parent_id) AS TablaAsociada,
    type_desc AS TipoTrigger,
    is_disabled AS Deshabilitado,
    create_date AS FechaCreacion
FROM sys.triggers
WHERE parent_id = OBJECT_ID('Clientes');
GO
```

3. Justificación técnica de la solución aplicada

Los triggers AFTER permiten auditoría automática y transparente sin modificar lógica de aplicación. Las tablas virtuales INSERTED y DELETED capturan estados antes/después de modificaciones. Almacenar valores anteriores y nuevos permite trazabilidad completa de cambios. SYSTEM_USER, HOST_NAME y APP_NAME identifican origen de modificaciones para investigaciones de seguridad. SET NOCOUNT ON en triggers evita interferencia con resultados de aplicaciones cliente.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- **Triggers AFTER**: No bloquean transacciones como INSTEAD OF
- **Tablas INSERTED/DELETED**: Capturan estados automáticamente
- **Auditoría completa**: Registra valores anteriores y nuevos
- **Metadatos contextuales**: Usuario, host, aplicación, fecha/hora
- **SET NOCOUNT ON**: Evita mensajes innecesarios en aplicaciones
- **Índices en auditoría**: Optimizan consultas históricas
- **Separación de triggers**: Un trigger por operación (claridad)
- **Nombres descriptivos**: TR_Tabla_Operacion_Proposito



Results Messages

	AuditoriaID	ClienteID	Operacion	Cliente	Usuario	FechaModificacion	HostName	NombreAplicacion
1	3	2	DELETE	María López Torres	Samirssj\homer	2025-11-06 15:30:56.810	SAMIRSSJ	Microsoft SQL Server Management Studio - Query
2	2	1	UPDATE	Juan Pérez García ? Juan Pérez García	Samirssj\homer	2025-11-06 15:30:56.790	SAMIRSSJ	Microsoft SQL Server Management Studio - Query
3	1	4	INSERT	Ana Fernández Ruiz	Samirssj\homer	2025-11-06 15:30:56.763	SAMIRSSJ	Microsoft SQL Server Management Studio - Query

	AuditoriaID	Operacion	CambioNombre	CambioEmail	Usuario	FechaModificacion
1	2	UPDATE	Nombre: Juan Pérez García ? Juan Pérez García	Email: juan.perez@email.com ? juan.perez.nuevo@e...	Samirssj\homer	2025-11-06 15:30:56.790

	NombreTrigger	TablaAsociada	TipoTrigger	Deshabilitado	FechaCreacion
1	TR_Clientes_Delete_Auditoria	Clientes	SQL_TRIGGER	0	2025-11-06 15:30:56.720
2	TR_Clientes_Update_Auditoria	Clientes	SQL_TRIGGER	0	2025-11-06 15:30:56.730
3	TR_Clientes_Insert_Auditoria	Clientes	SQL_TRIGGER	0	2025-11-06 15:30:56.747

Proyecto 10: Simulación de restauración tras un incidente en QhatuPeru

1. Enunciado del ejercicio

Por un error humano se eliminó información de la tabla Clientes. El equipo debe restaurar la base usando el respaldo más reciente y validar la recuperación exitosa de los datos.

Código T-SQL

```
-- PARTE A: Simulación del incidente

USE QhatuPeru;
GO

-- Verificar datos antes del "incidente"
SELECT 'ANTES DEL INCIDENTE' AS Momento, COUNT(*) AS TotalClientes
FROM Clientes;
GO

SELECT
    ClienteID,
    NombreCompleto,
    Email,
    Telefono
FROM Clientes
ORDER BY ClienteID;
GO
```

```
-- a) Simular la eliminación de registros de Clientes
BEGIN TRANSACTION;

-- Guardar count antes de eliminar
DECLARE @ClientesAntesIncidente INT;
SELECT @ClientesAntesIncidente = COUNT(*) FROM Clientes;

-- SIMULACIÓN DEL ERROR: Eliminación accidental
DELETE FROM Clientes;

-- Verificar que se eliminaron los registros
SELECT 'DESPUÉS DEL INCIDENTE' AS Momento, COUNT(*) AS TotalClientes
FROM Clientes;

PRINT 'INCIDENTE SIMULADO: Se eliminaron ' + CAST(@ClientesAntesIncidente AS
VARCHAR(10)) + ' registros de Clientes';

COMMIT TRANSACTION;
GO

-- Verificar el estado actual (tabla vacía)
SELECT COUNT(*) AS ClientesActuales FROM Clientes;
GO

-- PARTE B: Proceso de restauración

-- Paso 1: Realizar un respaldo TAIL-LOG antes de restaurar (captura
transacciones recientes)
USE master;
GO

-- Respaldo del tail del log (captura hasta el último momento)
BACKUP LOG QhatuPeru
TO DISK = 'C:\SQLBackups\QhatuPeru_TailLog.trn'
WITH NO_TRUNCATE, NORECOVERY;
GO

-- Paso 2: Restaurar desde el respaldo más reciente
-- Primero, verificar backups disponibles
RESTORE HEADERONLY
FROM DISK = 'C:\SQLBackups\QhatuPeru_Full.bak';
GO

-- Cerrar todas las conexiones activas a la base de datos
ALTER DATABASE QhatuPeru SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
GO

-- b) Restaurar desde el backup completo más reciente con NORECOVERY
RESTORE DATABASE QhatuPeru
FROM DISK = 'C:\SQLBackups\QhatuPeru_Full.bak'
WITH
    REPLACE,
```

```
    NORECOVERY,
    STATS = 10;
GO

-- Paso 3: Restaurar el tail-log backup si existe
RESTORE LOG QhatuPeru
FROM DISK = 'C:\SQLBackups\QhatuPeru_TailLog.trn'
WITH RECOVERY;
GO

-- Paso 4: Volver la base de datos a modo multi-usuario
ALTER DATABASE QhatuPeru SET MULTI_USER;
GO

-- VERIFICACIÓN DE LA RESTAURACIÓN

USE QhatuPeru;
GO

-- Verificar que los datos se restauraron correctamente
SELECT 'DESPUÉS DE LA RESTAURACIÓN' AS Momento, COUNT(*) AS TotalClientes
FROM Clientes;
GO

-- Consulta detallada para verificar integridad
SELECT
    ClienteID,
    NombreCompleto,
    Email,
    Telefono,
    FechaRegistro
FROM Clientes
ORDER BY ClienteID;
GO

-- Verificar la auditoría (debe mostrar las eliminaciones)
SELECT
    AuditoriaID,
    ClienteID,
    Operacion,
    NombreCompleto_Anterior,
    Usuario,
    FechaModificacion
FROM AuditoriaClientes
WHERE Operacion = 'DELETE'
ORDER BY FechaModificacion DESC;
GO

-- Script alternativo si no hay tail-log backup
-- (Restauración simple desde backup completo)
/*
USE master;
GO
```

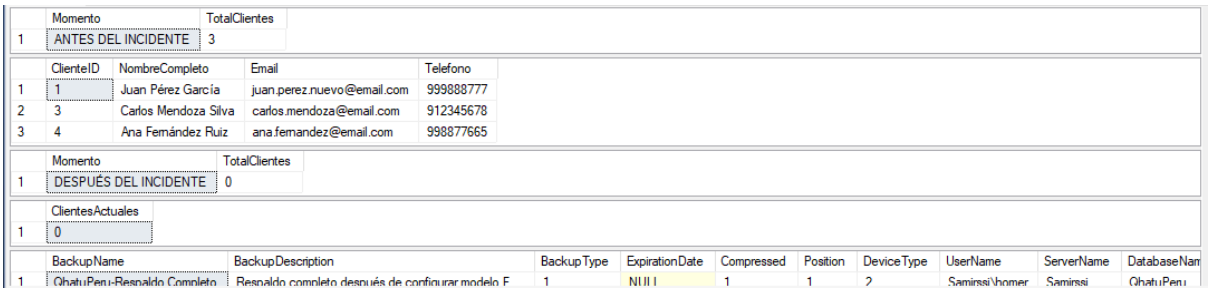
```
ALTER DATABASE QhatuPeru SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
GO

RESTORE DATABASE QhatuPeru
FROM DISK = 'C:\SQLBackups\QhatuPeru_Full.bak'
WITH
    REPLACE,
    RECOVERY,
    STATS = 10;
GO

ALTER DATABASE QhatuPeru SET MULTI_USER;
GO
*/

-- Generar reporte de restauración
SELECT
    'Restauración Completada' AS Estado,
    DB_NAME() AS BaseDatos,
    GETDATE() AS FechaRestauracion,
    (SELECT COUNT(*) FROM Clientes) AS ClientesRestaurados,
    (SELECT COUNT(*) FROM AuditoriaClientes WHERE Operacion = 'DELETE') AS
EliminacionesAuditadas;
GO

-- Consulta para verificar el historial de restauración
SELECT
    destination_database_name AS BaseDatos,
    restore_date AS FechaRestauracion,
    restore_type AS TipoRestauracion,
    user_name AS Usuario,
    CASE
        WHEN restore_type = 'D' THEN 'Database'
        WHEN restore_type = 'L' THEN 'Log'
        WHEN restore_type = 'F' THEN 'File'
        WHEN restore_type = 'I' THEN 'Differential'
    END AS DescripcionTipo
FROM msdb.dbo.restorehistory
WHERE destination_database_name = 'QhatuPeru'
ORDER BY restore_date DESC;
GO
```

3. Justificación técnica de la solución aplicada

La restauración con NORECOVERY permite aplicar múltiples respaldos secuencialmente (completo + diferenciales + logs) antes de la recuperación final. El tail-log backup captura transacciones comprometidas después del último respaldo, minimizando pérdida de datos. SINGLE_USER asegura que no hay conexiones activas durante restauración, evitando conflictos. La verificación post-restauración mediante COUNT y consultas detalladas valida integridad de datos. El historial en restorehistory permite auditar operaciones de recuperación.

4. Explicación de las buenas prácticas utilizadas en el proyecto

- **Tail-log backup**: Captura transacciones hasta el momento del incidente
- **NORECOVERY**: Permite aplicar múltiples respaldos secuencialmente
- **SINGLE_USER**: Previene conexiones durante restauración
- **REPLACE**: Sobrescribe base de datos existente
- **Verificación multi-nivel**: COUNT, consultas detalladas, auditoría
- **Documentación del proceso**: Comentarios explican cada paso
- **Consulta de historial**: restorehistory documenta operaciones
- **Scripts alternativos**: Proporciona opciones para diferentes escenarios
- **Reporte de restauración**: Resumen ejecutivo del proceso



	Momento	TotalClientes
1	ANTES DEL INCIDENTE	3

	ClienteID	NombreCompleto	Email	Telefono
1	1	Juan Pérez García	juan.perez.nuevo@email.com	999888777
2	3	Carlos Mendoza Silva	carlos.mendoza@email.com	912345678
3	4	Ana Fernández Ruiz	ana.fernandez@email.com	998877665

	Momento	TotalClientes
1	DESPUÉS DEL INCIDENTE	0

	ClientesActuales
1	0

	BackupName	BackupDescription	BackupType	ExpirationDate	Compressed	Position	DeviceType	UserName	ServerName	DatabaseNam
1	QhatuPeru-Respaldo Completo	Respaldo completo después de configurar modelo F	1	NULL	1	1	2	Samirssi\homer	Samirssi	QhatuPeru