



Universidad Nacional del Nordeste
Facultad de Ciencias Exactas y Naturales y Agrimensura
Licenciatura en Sistemas de Información
Base de Datos I

PROYECTO DE ESTUDIO:

“Sistema de Ventas de Insumos Informáticos – Implementación con Gestor de Base de Datos”

- **Profesores:**

- Darío Oscar, Villegas

- **Alumno:**

- Alan, Parras.
- Julio César, Pintos
- Giovanni Oscar, Piazza
- Mauricio Fernando, Ramirez Delgado.

BIBLIOGRAFIA.

| | |
|--|-----------|
| CAPÍTULO I: | 3 |
| Introducción | 3 |
| Definición del problema | 3 |
| Objetivo del Trabajo Práctico | 3 |
| Objetivos Generales | 3 |
| Objetivos Específicos | 4 |
| CAPÍTULO II: MARCO CONCEPTUAL | 5 |
| Definición de la Estructura del Dominio del Problema | 5 |
| Manejo de Permisos a Nivel de Usuarios de Bases de Datos | 6 |
| Procedimientos y Funciones Almacenadas | 8 |
| Optimización de Consultas a Través de Índices | 11 |
| Triggers | 12 |
| CAPÍTULO III: METODOLOGÍA SUGERIDA | 15 |
| Descripción de cómo se realizó el Trabajo Práctico | 15 |
| Herramientas | 15 |
| CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN DE RESULTADOS | 17 |
| Manejo de Permisos a Nivel de Usuarios de Bases de Datos | 17 |
| Procedimientos y Funciones Almacenadas | 23 |
| Optimización de Consultas a Través de Índices | 30 |
| Triggers | 33 |
| V. CONCLUSIONES | 37 |
| VI. BIBLIOGRAFÍA | 38 |

CAPÍTULO I: INTRODUCCIÓN

1.1 INTRODUCCIÓN

En el contexto actual, la gestión eficiente de bases de datos relacionales se ha convertido en un pilar fundamental para el desarrollo de sistemas de información robustos y seguros. A medida que las organizaciones manejan volúmenes crecientes de datos, surge la necesidad de implementar estrategias que no solo garanticen el acceso controlado y seguro a la información, sino también optimicen el rendimiento de las consultas y operaciones realizadas en las bases de datos.

1.2 Definición o planteamiento del problema.

En este proyecto se estudiará el **manejo de permisos a nivel de usuarios de base de datos**, destacando la importancia de establecer políticas de seguridad que definan de manera granular el acceso y la manipulación de datos.

También, se analizará el uso de **procedimientos y funciones almacenadas** como herramientas de encapsulamiento y reutilización de código SQL. Estas estructuras permiten la automatización de procesos, mejoran la consistencia de las operaciones y contribuyen a una mejor organización lógica dentro de la base de datos.

Luego, se abordará la **optimización de consultas a través de índices**, técnica que se enfoca en mejorar el rendimiento y la velocidad de las consultas SQL.

A lo largo de este proyecto, se realizará un análisis teórico y práctico de cada uno de estos temas, demostrando cómo su correcta aplicación puede contribuir significativamente a la eficiencia, seguridad y rendimiento de los sistemas de bases de datos relacionales en entornos académicos y profesionales.

Todo este análisis se llevará a cabo en base a un estudio de una **base de datos de un sistema de venta de insumos informáticos**.

1.3 Objetivo del Trabajo Práctico.

El presente proyecto tiene como objetivo explorar y aplicar conceptos avanzados de bases de datos relacionales, abordando tres temáticas claves: **manejo de permisos a nivel de usuarios de base de datos**, **procedimientos y funciones almacenadas** y **optimización de consultas a través de índices**.

1.3.1 Objetivos Generales.

Los objetivos generales es conseguir la correcta interpretación del problema, en base a su planteamiento teórico con los conceptos de la asignatura para su posterior implementación con software dedicado a la gestión de bases de datos, control de versionado, normalización y documentación del proyecto en cuestión, cumpliendo las propuestas

dictadas por la cátedra y ofreciendo una solución eficiente al problema.

1.3.2 Objetivos Específicos.

Identificar correctamente el DER asociado al problema con su posterior Diccionario de datos y scripts SQL competentes. Utilizar un esquema de niveles de usuarios por roles para brindar permisos y seguridad.

Mediante la creación de usuarios, implementar diferentes roles de seguridad en la base de datos para asignar permisos específicos respecto a cada rol.

Consultas SQL para testear el funcionamiento correcto y prevenir fallas o inconsistencias dentro del sistema.

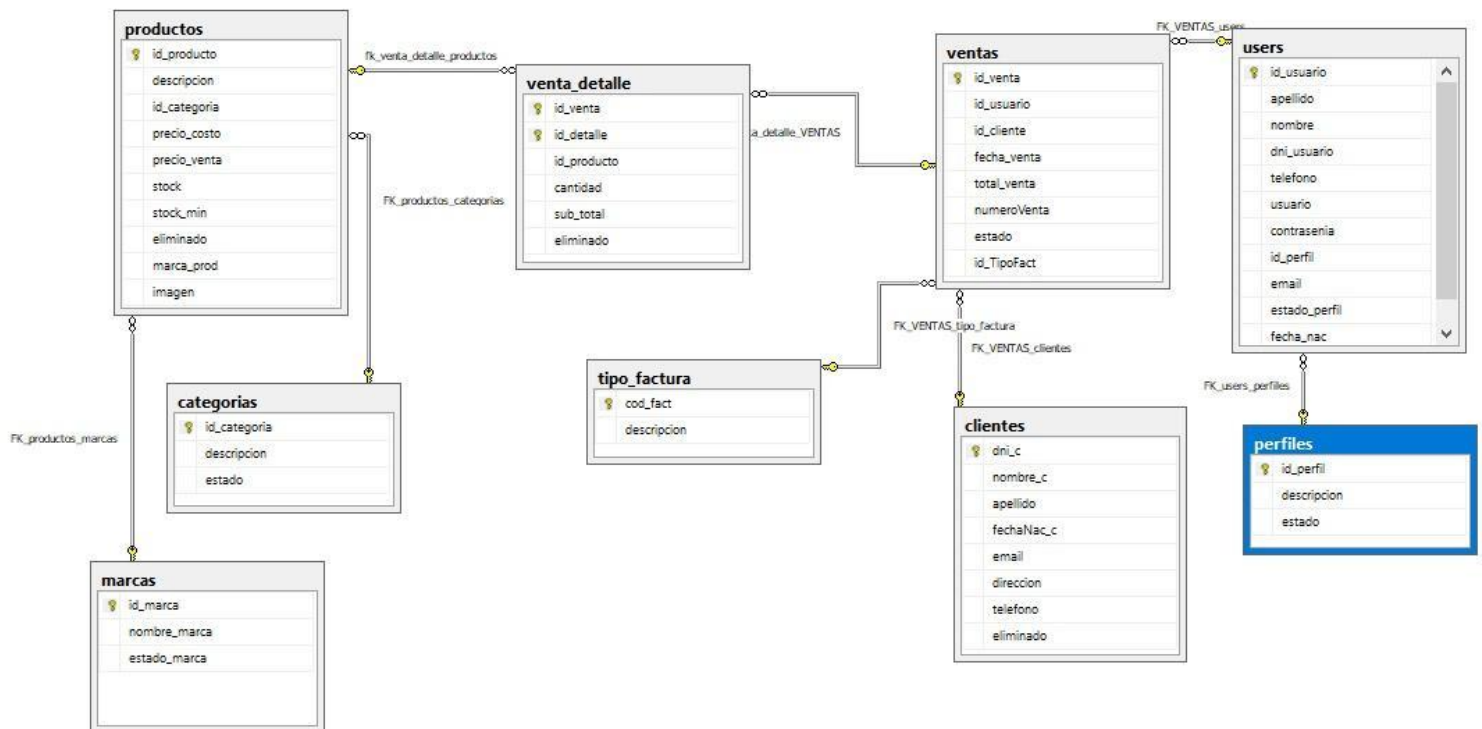
Utilización de índices y su posterior implementación para verificar su correcto desempeño y eficiencia en el manejo de la base de datos.

Correcta documentación y control de versionado del equipo en este caso de estudio para cumplimentar los objetivos.

CAPÍTULO II: MARCO CONCEPTUAL

2.1. Definición de la Estructura del Dominio del Problema.

El Diagrama de Entidad-Relación asociado al caso de estudio planteado es el siguiente:



Y su diccionario de datos asociado resulta, por ejemplo, en el caso de la Tabla Venta_Detalle*:

*la disposición completa se encuentra en el pdf adjunto en el repositorio

| Características de la Tabla | | | |
|------------------------------|--|------|--|
| Nombre | Venta_Detalle | | |
| Módulo | Ventas | | |
| Descripción | esta tabla guarda el detalle asociado a cada venta realizada | | |
| Características de los Datos | | | |
| Campo | Tipo | Long | Significado |
| id_venta | int | | identifica a la venta a la cual pertenece el detalle |
| id_detalle | int | - | identifica unitariamente al detalle mismo |
| id_producto | int | - | identifica al producto asociado al detalle |
| cantidad | int | - | establece la cantidad de dicho producto dentro del detalle |
| sub_total | float | - | calcula el total de la venta tomando la cantidad y la información según el id_producto |

| Restricciones | |
|-----------------|---------------------|
| Campo | Tipo de Restricción |
| id_detalle | PRIMARY KEY |
| Claves Foráneas | |
| Campo | Entidad Asociada |
| id_producto | Productos |
| id_venta | Ventas |

Sobre esta estructura relacional se llevarán a cabo las inserciones del lote de datos para su posterior implementación en SQL Server en la presentación de resultados de los tópicos ofrecidos. Dichos temas se encuentran desarrollados conceptualmente en los siguientes apartados (2.2 al 2.4), con sus consecuentes modelos de resultados en la sección Cuarta (4.1 al 4.4).

Empecemos pues, a definir los rasgos principales de cada uno de estos temas que nos guiarán en la materialización del caso de estudio.

2.2. Manejo de Permisos a Nivel de Usuarios de Bases de Datos.

Los permisos son los que nos permiten controlar qué acciones podemos realizar sobre los objetos de una base de datos, ya sean, tablas, vistas, también si podemos agregar o eliminar datos, entre otras cosas. Estos permisos se dividen en:

- Permisos de DML (Data Manipulation Language): son operaciones de consulta y modificación de datos.
 - *SELECT*: Permite consultar datos.
 - *INSERT*: Permite agregar nuevos registros.
 - *UPDATE*: Permite modificar registros existentes.
 - *DELETE*: Permite eliminar registros.
- Permisos de ejecución: Permiten a los usuarios ejecutar procedimientos almacenados y funciones:
 - *EXECUTE*: Permite la ejecución de procedimientos almacenados.
- Permisos de control de objetos: Permiten a los usuarios modificar o administrar objetos dentro de la base de datos:
 - *ALTER*: Permite modificar la estructura de un objeto.
 - *CONTROL*: Otorga el control total sobre un objeto.

SQL Server permite la creación y gestión de usuarios tanto a nivel de servidor como dentro de cada base de datos específica. La gestión de usuarios se realiza a través de la creación de cuentas y asignación de roles y permisos. Los usuarios pueden ser creados mediante el propio SQL Server o mediante autenticación de Windows.

Para crear un usuario se utilizan comandos como *CREATE LOGIN* para la autenticación de servidor y *CREATE USER* para la creación del usuario dentro de una base de datos específica.

Sintaxis de creación de un usuario mediante *LOGIN* y *USER*:

- Creación de usuario a nivel servidor
CREATE LOGIN alan WITH PASSWORD = '12345';

- Creación de usuario dentro de la base de datos proyecto_bd_I
USE proyecto_bd_I;
CREATE USER alan FOR LOGIN alan;

Los roles son colecciones de permisos que se pueden asignar a los usuarios para facilitar la administración. SQL Server ofrece dos tipos de roles:

- **Roles fijos de servidor:** Roles como sysadmin y securityadmin, que son definidos por SQL Server tienen permisos amplios a nivel de servidor.
- **Roles fijos de base de datos:** Roles como db_owner, db_datareader, y db_datawriter, que se aplican a nivel de base de datos para controlar operaciones específicas sobre los datos.

Sintaxis:

- asigna el rol db_owner al usuario
ALTER ROLE db_owner ADD MEMBER alan;
- crear un rol personalizado para analistas de datos
CREATE ROLE AnalistaDeDatos;

La gestión de permisos se realiza principalmente a través de los comandos:

- **GRANT:** Otorga un permiso a un usuario o rol.
- **DENY:** Niega un permiso específico a un usuario o rol, incluso si se le ha otorgado anteriormente.
- **REVOKE:** Elimina un permiso previamente otorgado o denegado.

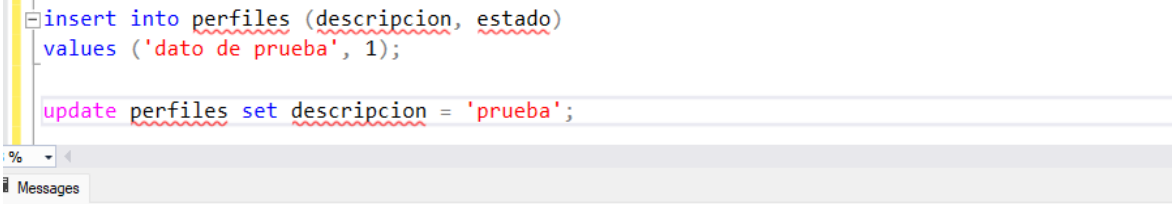
Ejemplo:

- Otorga permiso SELECT al usuario 'pepito' sobre una tabla productos

GRANT SELECT ON productos TO pepito;

- Deniega permiso INSERT al usuario 'pepito' sobre una tabla perfiles

DENY INSERT, UPDATE ON perfiles TO pepito



```

insert into perfiles (descripcion, estado)
values ('dato de prueba', 1);

update perfiles set descripcion = 'prueba';

```

Messages

Msg 229, Level 14, State 5, Line 32
The INSERT permission was denied on the object 'perfiles', database 'proyecto_bd_I', schema 'dbo'.
Msg 229, Level 14, State 5, Line 35
The UPDATE permission was denied on the object 'perfiles', database 'proyecto_bd_I', schema 'dbo'.

Completion time: 2024-11-12T00:34:02.2688441-03:00

El manejo adecuado de permisos en SQL Server es una de las estrategias fundamentales para proteger los datos y garantizar un acceso controlado. La implementación de roles, el uso del principio de menor privilegio y la revisión constante de permisos son prácticas esenciales para mantener la seguridad en un entorno de bases de datos relacionales. A través de herramientas como SSMS y mediante una correcta estructuración de permisos, los administradores pueden asegurar una gestión eficiente y segura de los datos.

2.3. Procedimientos y Funciones Almacenadas.

2.3.1. Rudimentos de Procedimientos Transact-SQL

Un procedimiento almacenado es un conjunto de instrucciones SQL que se almacena asociado a una base de datos. Es un objeto que se crea con la sentencia CREATE PROCEDURE y se invoca con la sentencia EXECUTE. Un procedimiento puede tener cero o muchos parámetros de entrada y cero o muchos parámetros de salida.

El lenguaje SQL Server consta de una o varias instrucciones Transact-SQL o una referencia a un método de CLR de Microsoft.NET Framework. Los procedimientos se asemejan a las construcciones de otros lenguajes de programación, porque pueden:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al programa que realiza la llamada
- Contener instrucciones de programación que realicen operaciones en la base de datos, como llamadas a otros procedimientos.
- Devolver un valor de estado a un programa que realiza una llamada para indicar que la operación se ha realizado correctamente o se han producido errores y cuál fue la causa.

Alguna de las ventajas que brinda el uso de Procedimientos son:

- Tráfico de Red Reducido entre el Cliente y el Servidor: los comandos de un procedimiento se ejecutan en un único lote de código, reduciendo significativamente el tráfico de red entre el servidor y el cliente porque sólo se envía a través de la red la llamada que va a ejecutar el procedimiento. Sin esta encapsulación del código, cada una de las líneas del script SQL tendría que enviarse a la red.
- Mayor Seguridad: varios usuarios y programas clientes pueden realizar operaciones en los objetos de base de datos subyacentes mediante un procedimiento, aunque los usuarios y los programas no tengan permisos y actividades se llevan a cabo y protegen los objetos de base de datos subyacentes. Esto elimina la necesidad de conceder permisos en cada nivel de objetos y simplifica los niveles de seguridad.
- Evitar Suplantaciones de Usuarios: la cláusula EXECUTE AS puede especificarse en la instrucción CREATE PROCEDURE para habilitar la suplantación de otro usuario o permitir que los usuarios o aplicaciones realicen ciertas actividades en la base de datos sin necesitar contar con permisos directos sobre objetos y comandos subyacentes.

Al llamar a un procedimiento en la red, solo está visible la llamada que va a ejecutar dicho procedimiento, por tanto los usuarios malintencionados no pueden ver los nombres de objetos y tablas de la DB incrustados en sus propias instrucciones Transact-SQL ni buscar datos críticos.

El uso de parámetros ayuda a protegerse contra ataques de inyección SQL, dado que la entrada de éstos se trata como valor literal y no como código ejecutable. El código de cualquier operación de DB redundante es candidato perfecto para la encapsulación de procedimientos, reduciendo inconsistencias y la necesidad de reescribir el código nuevamente.

De forma predeterminada, un procedimiento se compila la primera vez que se ejecuta y crea un plan de ejecución que vuelve a crearse en posteriores ejecuciones, necesitando así menos tiempo para procesar el procedimiento. Si hubo cambios importantes en las tablas o datos que referencia el procedimiento, el plan precompilado podría hacer que se ejecutará con mayor lentitud. Volver a crear el procedimiento y forzar un nuevo plan de ejecución mejoraría el rendimiento.

Existen diferentes tipos de Procedimientos Almacenados:

- **Definidos por el Usuario:** se puede crear en una DB definida por el usuario o en todas las DB del sistema, excepto en Resource. El procedimiento se puede desarrollar en Transact-SQL o como una referencia a un método de CLR de Microsoft.NET Framework.
- **Temporales:** forma de procedimientos definidos por el usuario similares a los permanentes, salvo porque se almacenan en tempdb. Existen 2 tipos: locales y globales. Se diferencian entre sí por nombres, visibilidad y disponibilidad. Los locales tienen como primer carácter de sus nombres un solo (#), solo son visibles en la conexión actual del usuario y se eliminan cuando esta se cierra. Los globales tienen dos signos (##), son visibles para cualquier usuario después de su creación, y se eliminan al final de la última sesión en la que se usa el procedimiento.
- **Sistema:** se incluyen con SQL Server, almacenados físicamente en la DB interna y oculta Resource, y se muestran de forma lógica en el esquema sys de cada DB definida por el sistema y el usuario. La DB msdb también contiene procedimientos almacenados del sistema en el esquema dbo que se usan para programar alertas y trabajos, dado que los procedimientos definidos del sistema empiezan con el prefijo sp_, se recomienda no usar esto cuando se asigne un nombre a los definidos por el usuario.

Los parámetros se usan para intercambiar datos entre funciones y procedimientos almacenados y la aplicación o herramienta que los llamo:

- ✓ Los parámetros de entrada permiten a quien realiza la llamada el pasar un valor de datos a la función o procedimiento almacenado.
- ✓ Los parámetros de salida permiten al procedimiento almacenado devolver un valor de datos o variable de cursor a quien realizó la llamada. Las funciones definidas por el usuario no pueden especificar parámetro de salida.
- ✓ Cada procedimiento almacenado devuelve un código de retorno de tipo entero a quien realiza la llamada. Si el procedimiento almacenado no establece explícitamente un valor para el código de retorno, éste es 0.

Para que un procedimiento almacenado admita parámetros de entrada se deben declarar variable como parámetros, Su sintaxis es:

```
CREATE PROCEDURE NOMBREPROCEDIMIENTO  
(@NOMBREPARAMETRO TIPO=VALORPORDEFECTO)  
AS  
SENTENCIAS;
```

Se aprecia como los parámetros se definen luego del nombre del procedimiento, con el signo @, y son locales al procedimiento mismo, solo existen dentro de este y pueden declararse varios de ellos, separándolos por coma. Para ejecutarlo, se utiliza el comando EXECUTE seguido del nombre

del procedimiento y los valores para cada parámetro.

2.3.2. Rudimentos de Funciones

Las funciones en SQL Server son objetos que se utilizan para realizar operaciones y obtener información. Existen 2 tipos:

- **Funciones del Sistema:** son aquellas que vienen incorporadas con SQL Server, como son las agregadas para valores numéricos, trabajar con fechas, etc. Ejemplos: SUM(), MAX(), DIFFDATE(), etc.
- **Funciones Definidas por el Usuario:** son establecidas por el usuario durante una sesión y pueden ser utilizadas en consultas.

Una función almacenada es un conjunto de instrucciones SQL que se almacena asociado a una base de datos. Es un objeto que se crea con la sentencia CREATE FUNCTION y se invoca con la sentencia SELECT o dentro de una expresión. Una función puede tener cero o muchos parámetros de entrada y siempre devuelve un valor, asociado al nombre de la función. Como se dijo arriba, una función definida por el usuario no puede asignar parámetros de salida.

Su sintaxis es:

```
CREATE FUNCTION NOMBREFUNCION
(NOMBREDELPARAMETRO TIPODELPARAMETRO)
RETURNS TIPODELRETORNO
AS
BEGIN
...
...RETURN ...
END
```

Los bloques BEGIN y END son obligatorios en una función, mientras que el procedimiento almacenado no los requiere si es solo una línea. En una función, es obligatorio utilizar los argumentos RETURNS y RETURN, mientras que en un procedimiento almacenado no es necesario.

En pocas palabras, un procedimiento almacenado es mucho más flexible para escribir cualquier código que uno desee, mientras que las funciones tienen una estructura y funcionalidad rígidas. Algunas características de la relación función-procedimiento:

- La principal ventaja de una función es que puede reutilizarse en código.
- Se puede concatenar fácilmente una función con una cadena. Para realizar algo similar con un procedimiento almacenado en SQL, vamos a necesitar una variable de salida en un procedimiento almacenado para poder concatenar la variable de salida con una cadena.
- Una ventaja de los procedimientos almacenados es que puede obtener varios parámetros mientras que, en las funciones, solo se puede devolver una variable (función escalar) o una tabla (funciones con valores de tabla).
- Es posible invocar funciones dentro de un procedimiento almacenado, pero no se puede invocar un procedimiento almacenado dentro de una función. También es posible invocar procedimientos desde un procedimiento.

2.4. Optimización de Consultas a Través de Índices.

En SQLSERVER, los índices son estructuras que se utilizan para mejorar la eficiencia de las consultas a una tabla. Estos índices contienen copias de los datos de la tabla, organizados de una manera que permite que las consultas encuentren los datos más rápidos.

Los índices son importantes en las bases de datos relacionales grandes, donde las consultas pueden ser muy complejas y los tiempos de respuestas de la consulta pueden ser críticos. Al agregar índices, se puede acelerar el proceso de recuperación de datos y mejorar el rendimiento de la base de datos.

Los tipos de índices que existen en SQL Server:

- **Clustered Index:** Es un tipo de índice que determina el orden físico de los datos en una tabla. Solo puede haber un índice agrupado por tabla y este ordena la tabla en función de la clave primaria. Es decir, los datos se almacenan en el disco en función de los valores de la columna de la clave primaria.

```
CREATE CLUSTERED INDEX IX_venta_numeroVenta ON [ventas]
([numeroVenta])
GO
```

Este ejemplo crea un clustered index en la columna **numeroVenta** de la tabla ventas, lo que significa que los registros se ordenarán físicamente en la tabla según el registro del **numeroVenta**.

- **Nonclustered Index:** A diferencia del índice agrupado, los índices no agrupados no ordenan físicamente la tabla. En su lugar, crean una estructura separada que incluye una copia de la columna de la clave primaria y la columna de índice. Esto permite una búsqueda más rápida de datos en la tabla.

```
CREATE NONCLUSTERED INDEX IX_ventas_numeroVenta
ON [ventas] (numeroVenta)
```

Este ejemplo crea un nonclustered index en la columna **numeroVenta** de la tabla ventas, lo que significa que se puede realizar una búsqueda rápida de registros según el número de venta de la venta registrada.

- **Unique Index:** Este tipo de índice se utiliza para garantizar que no se inserten valores duplicados en una tabla. Es similar a un índice no agrupado, pero solo puede haber un valor único para cada valor de la clave.

```
CREATE UNIQUE INDEX idx_productos_id_producto
ON productos (id_producto);
```

Este ejemplo crea un **unique index** en la columna **id_producto** de la tabla productos, lo que significa que no puede haber dos productos con el mismo código.

- **Filtered Index:** Este tipo de índice se utiliza para filtrar datos específicos en una tabla. Solo incluyen filas que cumplen con una condición específica. Esto reduce el tamaño del índice y mejora la velocidad de búsqueda de datos.

```
CREATE INDEX idx_ventas_recientes ON ventas (fecha_venta)
WHERE fecha_venta >= '2022-01-01';
```

Este ejemplo crea un filtered index en la columna **fecha_venta** de la tabla ventas, pero sólo para los pedidos realizados después del 1 de enero de 2022.

- **Full-Text Index:** Este tipo de índice se utiliza para buscar texto completo en una tabla. Permite la búsqueda de palabras clave y frases en lugar de simplemente buscar coincidencias exactas.

```
CREATE FULLTEXT INDEX ON productos (descripcion)
KEY INDEX idx_productos_id_producto;
```

Este ejemplo crea un full-text index en la columna **descripción** de la tabla productos, lo que significa que se puede realizar una búsqueda rápida de productos según la descripción. El index utiliza la columna id_producto como clave.

Es importante conocer los diferentes tipos de índices para poder elegir el más adecuado según las necesidades de la base de datos y mejorar la eficiencia de las consultas.

2.5 Triggers.

Un trigger es un procedimiento almacenado en la base de datos que se ejecuta automáticamente cada vez que ocurre un evento especial en la base de datos. Por ejemplo, un desencadenante puede activarse cuando se inserta una fila en una tabla específica o cuando ciertas columnas de la tabla se actualizan.

La creación de un disparador o trigger se realiza en dos pasos:

- En primer lugar, se crea la función disparadora.
- En segundo lugar, se crea el propio disparador SQL con el comando CREATE TRIGGER al que se introducen los parámetros para ejecutar la función disparadora.

Por lo general, estos eventos que desencadenan los triggers son cambios en las tablas mediante operaciones de inserción, eliminación y actualización de datos (insert, delete y update).

2.5.1 Eventos a los que responden los triggers:

- **INSERT:** El *trigger* se activa cuando se inserta una nueva fila sobre la tabla asociada.
- **UPDATE:** El *trigger* se activa cuando se actualiza una fila sobre la tabla asociada.
- **DELETE:** El *trigger* se activa cuando se elimina una fila sobre la tabla asociada.

2.5.2 Clases de Triggers en SQL:

- **Triggers DDL (Data Definition Language):** Esta clase de Triggers se activa en eventos que modifican la estructura de la base de datos (como crear, modificar o eliminar una tabla) o en ciertos eventos relacionados con el servidor, como cambios de seguridad o actualización de eventos estadísticos.
- **Triggers DML (Data Modification Language):** Esta es la clase más común de Triggers. En este caso, el evento de disparo es una declaración de modificación de datos; podría ser una declaración de inserción, actualización o eliminación en una tabla o vista.

A su vez estos últimos tienen diferentes tipos:

- **FOR o AFTER [INSERT, UPDATE, DELETE]:** Estos tipos de Triggers se ejecutan después de completar la instrucción de disparo (inserción, actualización o eliminación).
- **INSTEAD OF [INSERT, UPDATE, DELETE]:** A diferencia del tipo **FOR (AFTER)**, los Triggers **INSTEAD OF** se ejecutan en lugar de la instrucción de disparo. En otras palabras, este tipo de trigger reemplaza la instrucción de disparo. Son de gran utilidad en los casos en los que es necesario tener integridad referencial entre bases de datos.

Ventajas:

- Generar automáticamente algunos valores de columna derivados.
- Aplicar la integridad referencial.
- Registro de eventos y almacenamiento de información sobre el acceso a la tabla.
- Auditoría.
- Replicación sincrónica de tablas.
- Imponer autorizaciones de seguridad.
- Prevenir transacciones inválidas.

Utilización:

CREATE

```
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
[trigger_order]
trigger_body
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

Ejemplo:

Trigger para evitar ventas con productos fuera de stock

Este trigger verifica si el stock del producto es suficiente antes de permitir una venta

```
CREATE TRIGGER trg_check_stock
```

```
ON [dbo].[venta_detalle]
```

```
INSTEAD OF INSERT
```

```
AS
```

```
BEGIN
```

```
    SET NOCOUNT ON;
```

-- Verificar si el stock es suficiente para cada producto en la venta

```
    IF EXISTS (
```

```
        SELECT vd.id_producto, vd.cantidad, p.stock
```

```
        FROM inserted vd
```

```
        INNER JOIN productos p ON vd.id_producto = p.id_producto
```

```
        WHERE vd.cantidad > p.stock )
```

```
    BEGIN
```

```
        RAISERROR ('No hay suficiente stock para uno o más productos.', 16, 1);
```

```
        ROLLBACK TRANSACTION;
```

```
    END
```

```
    ELSE
```

```
    BEGIN
```

-- Insertar el detalle de la venta si el stock es suficiente

```
        INSERT INTO venta_detalle (id_venta, id_producto, cantidad, sub_total,
eliminado)
```

```
        SELECT id_venta, id_producto, cantidad, sub_total, eliminado
```

```
        FROM inserted;
```

-- Actualizar el stock de los productos

```
        UPDATE p
```

```
        SET p.stock = p.stock - vd.cantidad
```

```
        FROM productos p
```

```
        INNER JOIN inserted vd ON p.id_producto = vd.id_producto;
```

```
    END
```

```
END;
```

```
GO
```

CAPÍTULO III: METODOLOGÍA SUGERIDA

3.1 Descripción de cómo se realizó el Trabajo Práctico:

Para iniciar el desarrollo, planificamos y coordinamos el proyecto mediante una reunión a través de la plataforma Discord. Durante esta sesión, discutimos y trazamos un plan de acción que nos serviría como hoja de ruta para el trabajo.

Tras esta fase inicial, comenzamos con la investigación sobre los temas a desarrollar en el proyecto. Cada uno de los participantes se dedicó a investigar de manera individual un tema, explorando diversas fuentes de información, documentos relevantes y estudios relacionados lo que nos permitió obtener una comprensión más profunda del tema en cuestión.

Una vez que contamos con los conocimientos necesarios, procedimos a la fase de implementación práctica de nuestro proyecto. En este punto, cada miembro del equipo se centró en desarrollar código y herramientas de manera individual, lo que nos brindó la oportunidad de sumergirnos en los detalles técnicos y entender a fondo las complejidades de lo que estábamos desarrollando. Esta aproximación individual permitió a cada uno de nosotros adquirir conocimiento de la implementación, lo que a su vez contribuyó a la calidad y robustez del proyecto en su conjunto.

Finalmente, una vez que cada miembro había completado su contribución, unificamos nuestro trabajo y conocimientos para crear una solución coherente.

3.2 Herramientas (Instrumentos y procedimientos)

3.2.1 *Discord (Comunicaciones grupales)*

- Creamos un servidor en Discord para trabajo.
- Establecimos un canal específico para la realización y actualización del proyecto.
- Por último, utilizamos las llamadas de Discord para discusiones generales, actualizaciones rápidas y comunicación en tiempo real.

3.2.2 *WhatsApp (Seguimiento de avances):*

- Creamos un grupo de WhatsApp para el equipo.
- En este se compartió actualizaciones diarias sobre avances.

3.2.3 *GitHub (Gestión de código fuente):*

- Creamos un repositorio en GitHub para el proyecto.
- Utilizamos ramas para trabajar en funciones o características específicas.
- Fuimos subiendo commits con los avances en el proyecto.

3.2.4 SQL Server Management Studio (Administración de bases de datos):

- Utilizamos SQL Server Management Studio para diseñar, desarrollar y administrar bases de datos.
- Se emplea para realizar consultas SQL para extraer información relevante.

3.2.5 Google Docs (Colaboración en documentación):

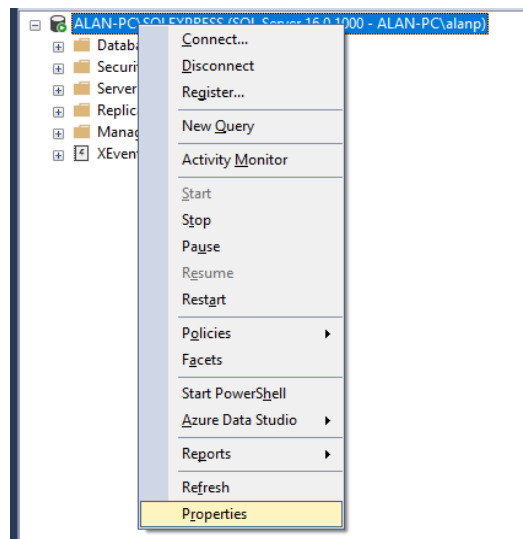
- Creamos un documento compartido en donde vamos recopilando la información y redacción final del informe.
- Nos permite y facilita la colaboración en tiempo real para la redacción y documentación del informe.

CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN DE RESULTADOS

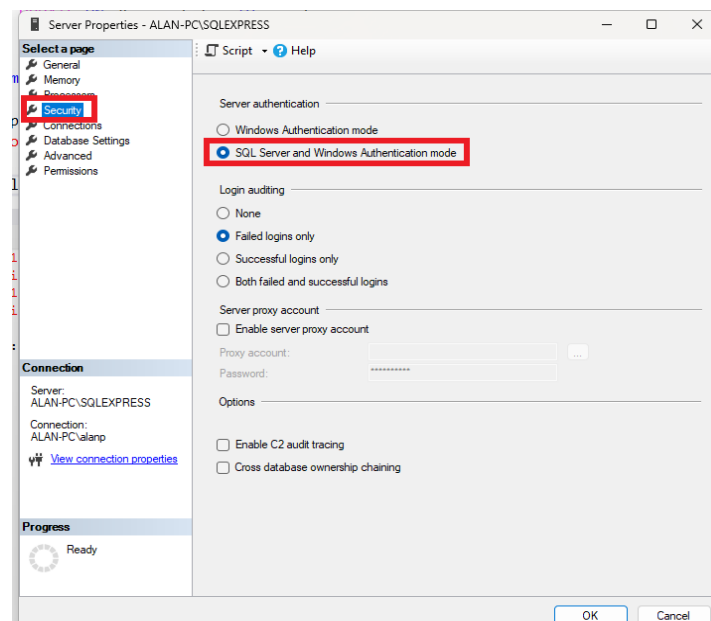
4.1. Manejo de Permisos a Nivel de Usuarios de Bases de Datos.

Para poder cambiar de usuario primero debemos verificar que el modo de autenticación de SQL Server este configurado en modo mixto (permitiendo autenticación de SQL Server y autenticación de Windows), para ello se debe hacer lo siguiente:

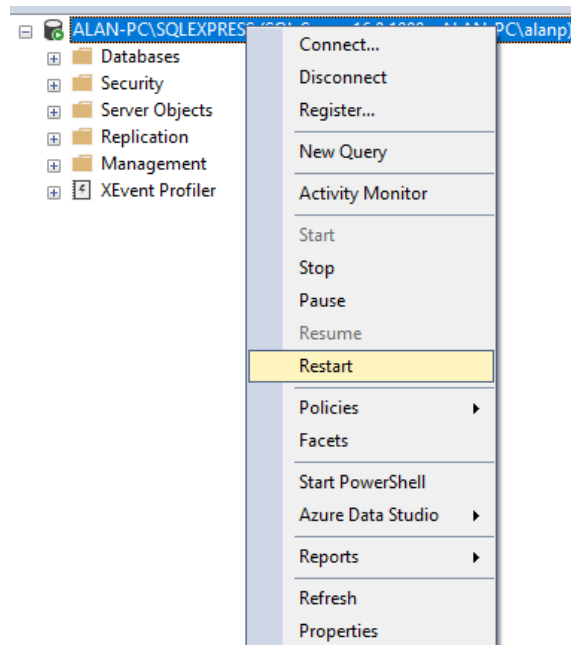
Clic derecho sobre el nombre del servidor (en mi caso, ALAN-PC\SQLEXPRESS), y seleccionar “Propiedades”.



En la ventana de Propiedades del servidor vamos a la sección “Seguridad”, y en Autenticación del Servidor seleccionaremos el Modo de autenticación de SQL Server y Windows, luego daremos clic en Ok para guardar los cambios.



Por último debemos reiniciar el servidor de SQL Server, esto lo hacemos dándole clic derecho al nombre del servidor y seleccionando “Reiniciar”.



4.1.1. Prueba de Manejo de Permisos a Nivel de Roles

Primero se crean un par de usuarios para realizar las pruebas correspondientes, luego se le asignan los roles y permisos específicos para cada uno de ellos.

```
use proyecto_bd_I;

----- (1) Permisos a nivel de usuarios -----

-- creacion de UsuarioAdmin y UsuarioLectura a través de LOGIN
CREATE LOGIN UsuarioAdmin WITH PASSWORD = '1234';
CREATE LOGIN UsuarioLectura WITH PASSWORD = '1234';

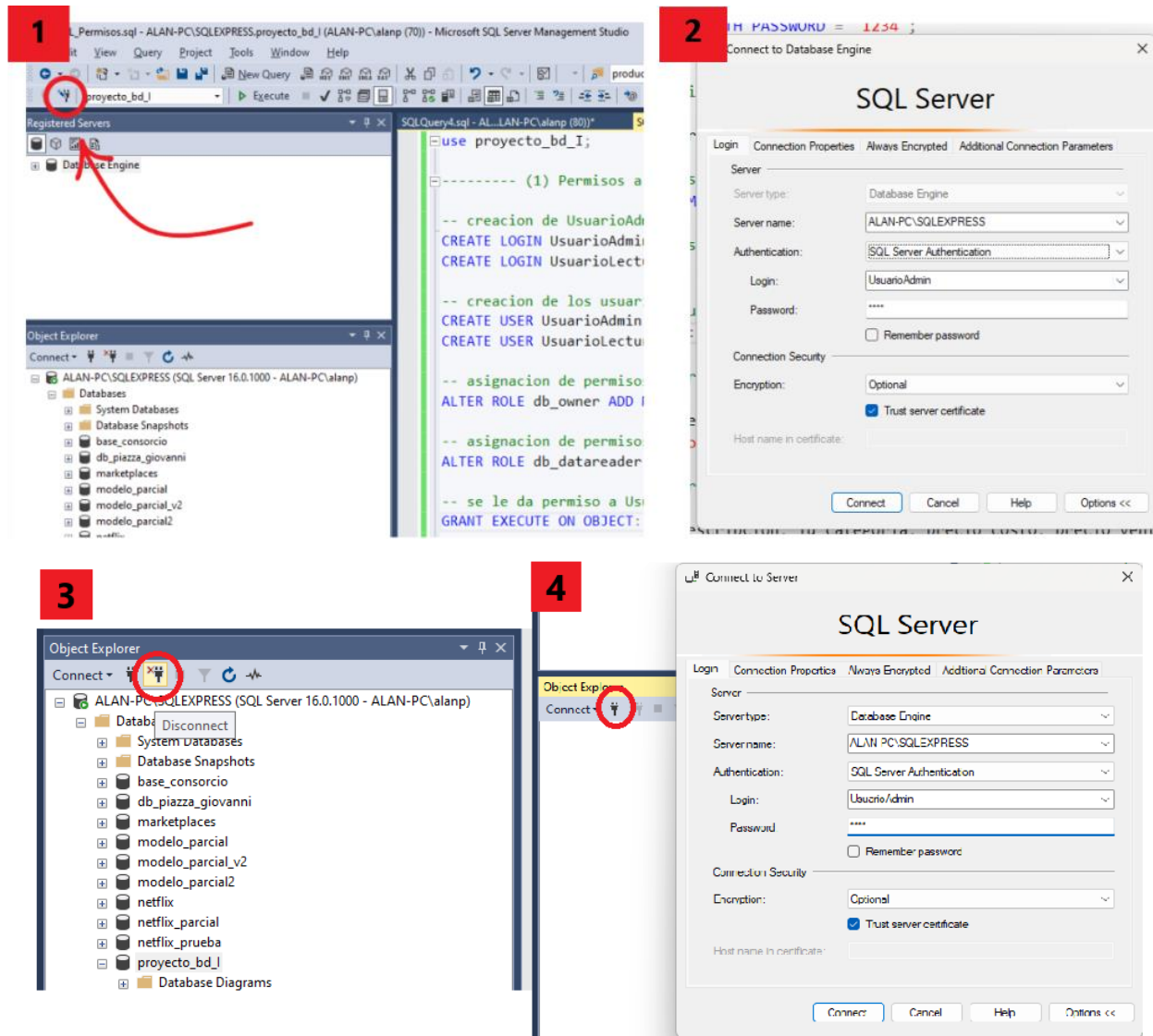
-- creacion de los usuarios en la base de datos
CREATE USER UsuarioAdmin FOR LOGIN UsuarioAdmin;
CREATE USER UsuarioLectura FOR LOGIN UsuarioLectura;

-- asignacion de permisos de administrador
ALTER ROLE db_owner ADD MEMBER UsuarioAdmin;

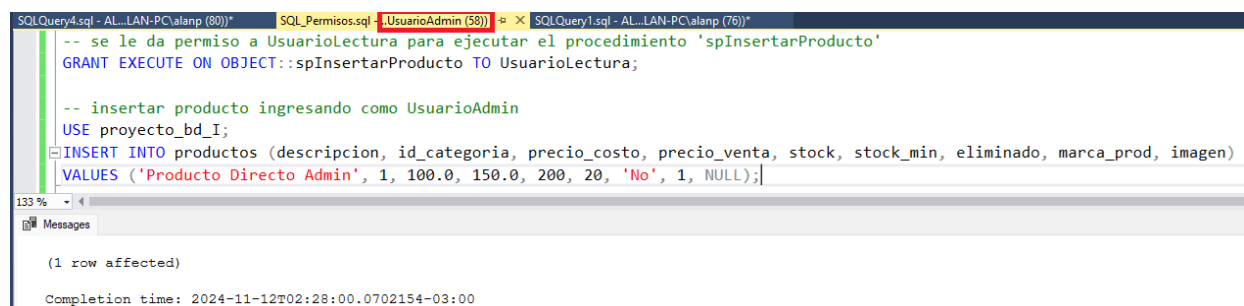
-- asignacion de permisos de lectura
ALTER ROLE db_datareader ADD MEMBER UsuarioLectura;

-- se le da permiso a UsuarioLectura para ejecutar el procedimiento 'spInsertarProducto'
GRANT EXECUTE ON OBJECT::dbo.spInsertarProducto TO UsuarioLectura;
```

A continuación nos conectamos como UsuarioAdmin para realizar la inserción de datos.

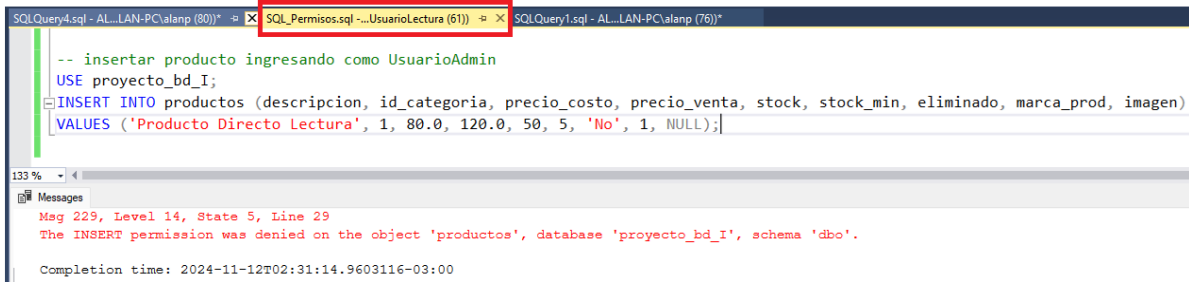


Luego procedemos a insertar.



Repetimos el proceso pero conectándonos como "UsuarioLectura".

Como podemos ver las restricciones que se aplicaron sobre “UsuarioLectura” no le permite ingresar productos.

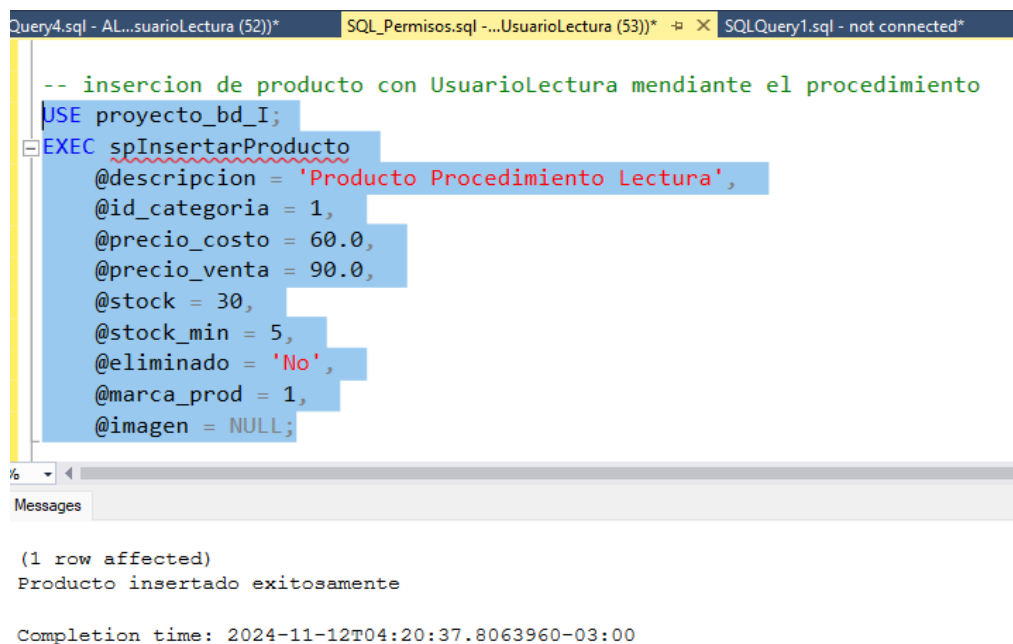


```
-- insertar producto ingresando como UsuarioAdmin
USE proyecto_bd_I;
INSERT INTO productos (descripcion, id_categoria, precio_costo, precio_venta, stock, stock_min, eliminado, marca_prod, imagen)
VALUES ('Producto Directo Lectura', 1, 80.0, 120.0, 50, 5, 'No', 1, NULL);
```

Msg 229, Level 14, State 5, Line 29
The INSERT permission was denied on the object 'productos', database 'proyecto_bd_I', schema 'dbo'.

Completion time: 2024-11-12T02:31:14.9603116-03:00

Ahora probamos insertar un producto a través del procedimiento.



```
-- insercion de producto con UsuarioLectura mediante el procedimiento
USE proyecto_bd_I;
EXEC spInsertarProducto
    @descripcion = 'Producto Procedimiento Lectura',
    @id_categoria = 1,
    @precio_costo = 60.0,
    @precio_venta = 90.0,
    @stock = 30,
    @stock_min = 5,
    @eliminado = 'No',
    @marca_prod = 1,
    @imagen = NULL;
```

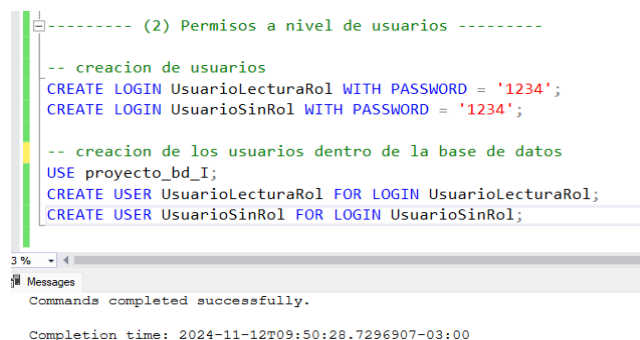
(1 row affected)
Producto insertado exitosamente

Completion time: 2024-11-12T04:20:37.8063960-03:00

Lo cual se completa exitosamente, ya que UsuarioLectura tiene permisos de ejecución sobre “spInsertarProducto”, aunque no pueda hacer un INSERT directo en la tabla.

4.1.2. Prueba de Manejo de Permisos a nivel de roles del DBMS

Para realizar esta prueba crearemos 2 nuevos usuarios.



```
----- (2) Permisos a nivel de usuarios -----
-- creacion de usuarios
CREATE LOGIN UsuarioLecturaRol WITH PASSWORD = '1234';
CREATE LOGIN UsuarioSinRol WITH PASSWORD = '1234';

-- creacion de los usuarios dentro de la base de datos
USE proyecto_bd_I;
CREATE USER UsuarioLecturaRol FOR LOGIN UsuarioLecturaRol;
CREATE USER UsuarioSinRol FOR LOGIN UsuarioSinRol;
```

Commands completed successfully.

Completion time: 2024-11-12T09:50:28.7296907-03:00

Creamos un rol que permita leer solamente la tabla de clientes, y se lo asignamos a “UsuarioLecturaRol”.

```
-- creacion del rol
CREATE ROLE RolSoloLecturaProductos;

-- otorgamos permisos de lectura sobre la tabla 'productos' al rol
GRANT SELECT ON dbo.productos TO RolSoloLecturaProductos;

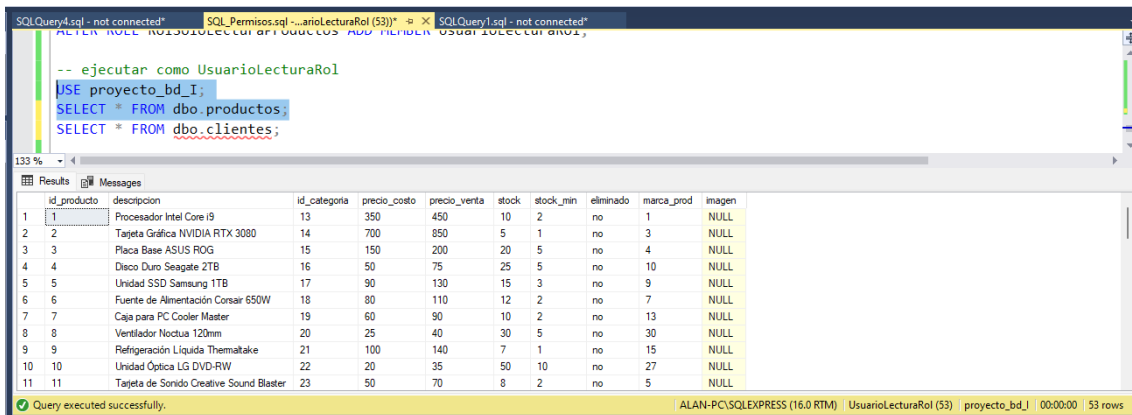
-- asigna el rol de solo lectura al usuario UsuarioLecturaRol
ALTER ROLE RolSoloLecturaProductos ADD MEMBER UsuarioLecturaRol;
```

Commands completed successfully.

Completion time: 2024-11-12T09:57:39.3363593-03:00

A continuación verificaremos el comportamiento de ambos usuarios.

Primero nos conectamos como “UsuarioLecturaRol” y probamos leer algunas tablas, entre ellas la de productos.

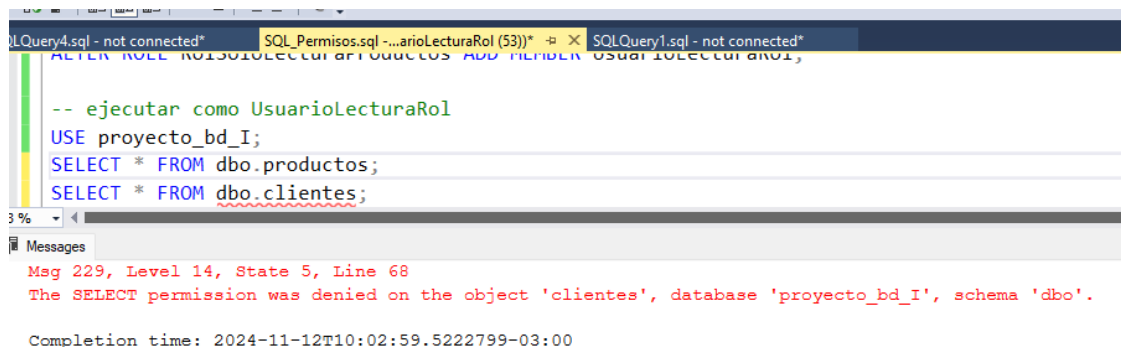


```
-- ejecutar como UsuarioLecturaRol
USE proyecto_bd_I;
SELECT * FROM dbo.productos;
SELECT * FROM dbo.clientes;
```

| id_producto | descripcion | id_categoria | precio_costo | precio_venta | stock | stock_min | eliminado | marca_prod | imagen |
|-------------|--|--------------|--------------|--------------|-------|-----------|-----------|------------|--------|
| 1 | Procesador Intel Core i9 | 13 | 350 | 450 | 10 | 2 | no | 1 | NULL |
| 2 | Tarjeta Gráfica NVIDIA RTX 3080 | 14 | 700 | 850 | 5 | 1 | no | 3 | NULL |
| 3 | Placa Base ASUS ROG | 15 | 150 | 200 | 20 | 5 | no | 4 | NULL |
| 4 | Disco Duro Seagate 2TB | 16 | 50 | 75 | 25 | 5 | no | 10 | NULL |
| 5 | Unidad SSD Samsung 1TB | 17 | 90 | 130 | 15 | 3 | no | 9 | NULL |
| 6 | Fuente de Alimentación Corsair 650W | 18 | 80 | 110 | 12 | 2 | no | 7 | NULL |
| 7 | Caja para PC Cooler Master | 19 | 60 | 90 | 10 | 2 | no | 13 | NULL |
| 8 | Ventilador Noctua 120mm | 20 | 25 | 40 | 30 | 5 | no | 30 | NULL |
| 9 | Refrigeración Líquida Thermaltake | 21 | 100 | 140 | 7 | 1 | no | 15 | NULL |
| 10 | Unidad Óptica LG DVD-RW | 22 | 20 | 35 | 50 | 10 | no | 27 | NULL |
| 11 | Tarjeta de Sonido Creative Sound Blaster | 23 | 50 | 70 | 8 | 2 | no | 5 | NULL |

Query executed successfully. ALAN-PC\SQLEXPRESS (16.0 RTM) | UsuarioLecturaRol (53) | proyecto_bd_I | 00:00:00 | 53 rows

Intentamos leer otra tabla, en este caso es la de clientes, y vemos que no nos lo permite.

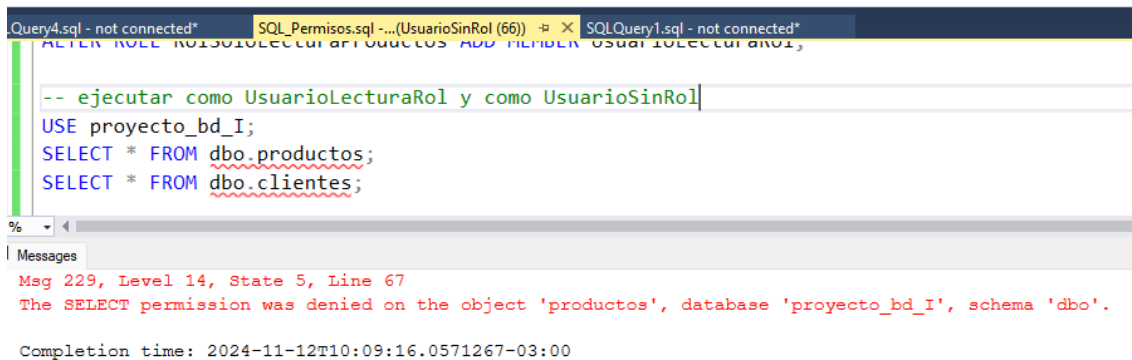


```
-- ejecutar como UsuarioLecturaRol
USE proyecto_bd_I;
SELECT * FROM dbo.productos;
SELECT * FROM dbo.clientes;
```

Msg 229, Level 14, State 5, Line 68
The SELECT permission was denied on the object 'clientes', database 'proyecto_bd_I', schema 'dbo'.

Completion time: 2024-11-12T10:02:59.5222799-03:00

Ahora intentaremos leer nuevamente algunas tablas, pero esta vez nos conectaremos como UsuarioSinRol.

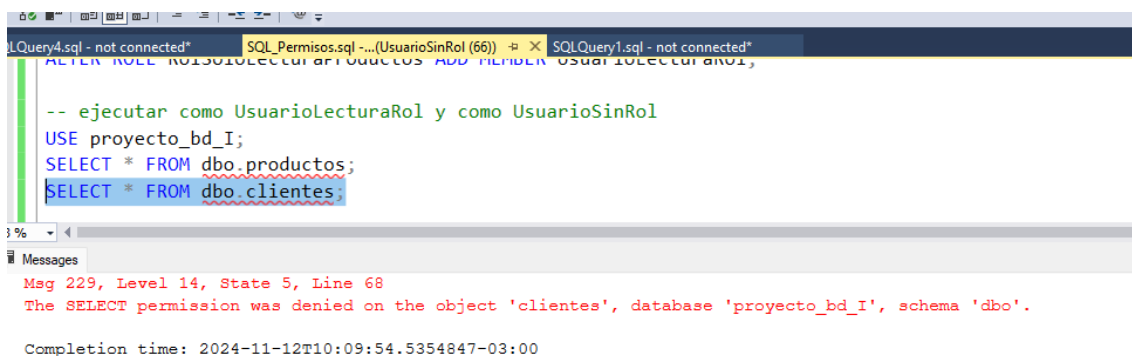


The screenshot shows a SQL Server Enterprise Manager window with a query editor. The query is as follows:

```
ALTER ROLE rolUsuarioLecturaProductos ADD MEMBER UsuarioLecturaRol1,  
  
-- ejecutar como UsuarioLecturaRol y como UsuarioSinRol  
USE proyecto_bd_I;  
SELECT * FROM dbo.productos;  
SELECT * FROM dbo.clientes;
```

The Messages pane at the bottom shows the following error:

```
Msg 229, Level 14, State 5, Line 67  
The SELECT permission was denied on the object 'productos', database 'proyecto_bd_I', schema 'dbo'.  
  
Completion time: 2024-11-12T10:09:16.0571267-03:00
```



The screenshot shows the same SQL Server Enterprise Manager window with the same query. The Messages pane now shows a second error:

```
Msg 229, Level 14, State 5, Line 68  
The SELECT permission was denied on the object 'clientes', database 'proyecto_bd_I', schema 'dbo'.  
  
Completion time: 2024-11-12T10:09:54.5354847-03:00
```

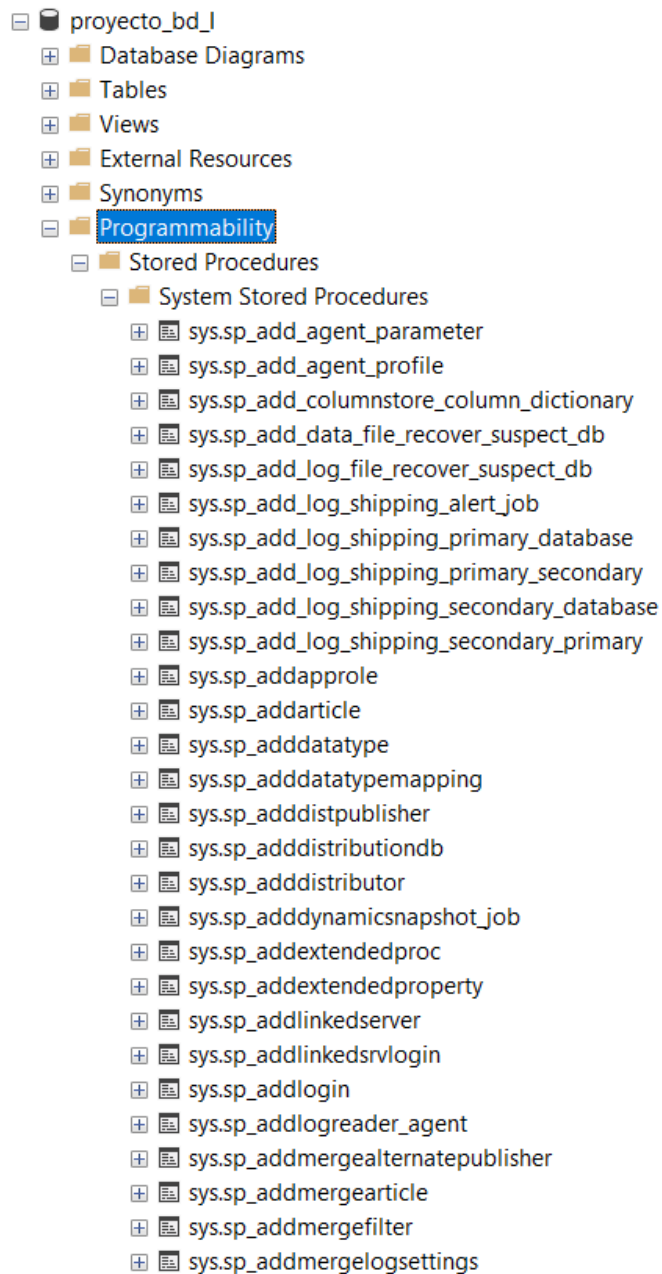
Luego de la implementación de este ejercicio, llegamos a la conclusión de que el uso de roles para asignar permisos específicos, como en este caso, en el que asignamos permisos de SELECT sobre una tabla en particular, facilita la administración de accesos para múltiples usuarios. Esto se debe a que podemos añadir o remover usuarios del rol sin tener que ajustar los permisos individualmente, lo cual ahorra tiempo y reduce el riesgo de errores administrativos en la asignación de permisos.

4.2. Procedimientos y Funciones Almacenadas.

Referenciando a lo mostrado en 2.3. sobre Procedimientos y Funciones, el script SQL del Tema02 contiene el siguiente desarrollo.

4.2.1. Procedimientos Almacenados del Sistema

Estos pueden verse dentro de la subcarpeta Programmability -> Stored Procedures -> System Stored Procedures.



4.2.2. Procedimientos Definidos por el Usuario

Primeramente, una función equivalente al INSERT, en este caso para añadir registros a la tabla productos.

```
--1. Procedimiento para Insertar elementos
CREATE PROCEDURE spInsertarProducto
    @descripcion VARCHAR(50),
    @id_categoria INT,
    @precio_costo FLOAT,
    @precio_venta FLOAT,
    @stock INT,
    @stock_min INT,
    @eliminado VARCHAR(30),
    @marca_prod INT = NULL, -- Se permite NULL para marca_prod
    @imagen VARCHAR(MAX) = NULL
AS
BEGIN
    INSERT INTO productos (descripcion, id_categoria, precio_costo, precio_venta, stock, stock_min, eliminado, marca_prod, imagen)
    VALUES (@descripcion, @id_categoria, @precio_costo, @precio_venta, @stock, @stock_min, @eliminado, @marca_prod, @imagen);

    PRINT 'Producto insertado exitosamente';
END;

EXEC spInsertarProducto 'Producto Procedimiento', 1, 50.00, 70.00, 100, 10, 'No', 1, NULL;
```

Ahora, una función equivalente al DELETE para borrar un producto según su descripción.

```
--2. Procedimiento para Delete elemento por nombre
CREATE PROCEDURE spEliminarProducto
    @descripcion VARCHAR(50)
AS
BEGIN
    DELETE FROM productos
    WHERE descripcion = @descripcion;

    PRINT 'Producto eliminado exitosamente';
END;

EXEC spEliminarProducto 'Producto Procedimiento'; --eliminando el producto que creamos
```

Y un procedimiento que nos será de utilidad, permitiendo alterar en igual grado los precios de una misma marca de productos.

```
--3. Procedimiento para modificar todos los precios de productos una misma marca
CREATE PROCEDURE spActualizarPrecioVentaMarca(
    @NombreMarca VARCHAR(30),
    @PrecioNuevo DECIMAL(6,2)
)
AS
BEGIN
    -- Actualizar el precio de venta de los productos de la marca especificada
    UPDATE p
    SET p.precio_venta = p.precio_venta * @PrecioNuevo
    FROM productos p
    INNER JOIN marcas m ON p.marca_prod = m.id_marca
    WHERE m.nombre_marca = @NombreMarca

    PRINT 'Precios actualizados correctamente.'
END;

EXECUTE spActualizarPrecioVentaMarca 'Intel', 1.20 --aumenta el precio en un 20%
```


4.2.2.1. Insertar Datos en el Modelo

Primero, se realizó la inserción usando INSERT y este fue el resultado.

```
--set IDENTITY_INSERT productos on;
--Inserción de Datos con Insert
INSERT INTO productos (id_producto, descripcion, id_categoria, precio_costo, precio_venta, stock, stock_min, eliminado, marca_prod, imagen)
VALUES
(51, 'Laptop ASUS VivoBook 15', 1, 500.00, 650.00, 15, 5, 'no', 4, NULL),
(52, 'Monitor ASUS 24" Full HD', 2, 150.00, 200.00, 30, 10, 'no', 4, NULL),
(53, 'Teclado ASUS Gaming', 3, 50.00, 75.00, 40, 15, 'no', 4, NULL),
(54, 'Ratón ASUS Gaming', 3, 30.00, 50.00, 35, 10, 'no', 4, NULL),
(55, 'Laptop ASUS ROG Strix', 1, 1000.00, 1300.00, 20, 5, 'no', 4, NULL),
(56, 'Auriculares ASUS ROG', 3, 70.00, 100.00, 50, 20, 'no', 4, NULL),
(57, 'SSD ASUS 500GB', 2, 80.00, 120.00, 25, 10, 'no', 4, NULL),
(58, 'Memoria RAM ASUS 16GB', 2, 60.00, 90.00, 40, 15, 'no', 4, NULL),
(59, 'Laptop ASUS ZenBook', 1, 750.00, 950.00, 18, 5, 'no', 4, NULL),
(60, 'Monitor ASUS 27" 4K', 2, 300.00, 450.00, 12, 5, 'no', 4, NULL),
(61, 'Mousepad ASUS RGB', 3, 20.00, 35.00, 50, 20, 'no', 4, NULL),
(62, 'Laptop ASUS TUF Gaming', 1, 800.00, 1050.00, 22, 7, 'no', 4, NULL),
(63, 'Mouse ASUS TUF', 3, 40.00, 60.00, 38, 12, 'no', 4, NULL),
(64, 'Teclado ASUS TUF', 3, 60.00, 85.00, 30, 12, 'no', 4, NULL),
(65, 'Adaptador ASUS WiFi', 2, 15.00, 25.00, 60, 25, 'no', 4, NULL),
(66, 'Laptop ASUS ExpertBook', 1, 950.00, 1250.00, 19, 6, 'no', 4, NULL),
(67, 'Cargador ASUS Laptop', 2, 30.00, 45.00, 40, 15, 'no', 4, NULL),
(68, 'Mochila ASUS Gaming', 3, 40.00, 60.00, 25, 10, 'no', 4, NULL),
(69, 'Pantalla ASUS 32" Curva', 2, 400.00, 550.00, 15, 5, 'no', 4, NULL),
(70, 'Cable HDMI ASUS', 2, 10.00, 20.00, 50, 20, 'no', 4, NULL),
(71, 'Laptop ASUS X515', 1, 400.00, 550.00, 25, 10, 'no', 4, NULL),
(72, 'Pantalla ASUS 24" Full HD', 2, 140.00, 180.00, 30, 12, 'no', 4, NULL),
(73, 'Teclado ASUS Wireless', 3, 45.00, 70.00, 50, 20, 'no', 4, NULL),
(74, 'Laptop ASUS Flip 14', 1, 650.00, 850.00, 20, 8, 'no', 4, NULL),
(75, 'Monitor ASUS 34" Curvo', 2, 500.00, 700.00, 12, 5, 'no', 4, NULL),
(76, 'Disco Duro ASUS 1TB', 2, 90.00, 130.00, 20, 8, 'no', 4, NULL),
(77, 'Mochila ASUS 15"', 3, 30.00, 45.00, 35, 12, 'no', 4, NULL),
(78, 'Auriculares ASUS 7.1', 3, 100.00, 130.00, 25, 10, 'no', 4, NULL),
(79, 'Laptop ASUS TUF Dash', 1, 950.00, 1200.00, 20, 7, 'no', 4, NULL),
(80, 'Teclado ASUS RGB', 3, 55.00, 80.00, 45, 15, 'no', 4, NULL),
```

99 %

Messages

(40 rows affected)

Completion time: 2024-11-11T10:41:32.8163630-03:00

99 %

Query executed successfully. DESKTOP-BQ97A1E\SQLEXPRESS ... DESKTOP-BQ97A1E\Usuari... proyecto_bd_J 00:00:00 0 rows

Luego, se insertaron más elementos utilizando el procedimiento creado (cuyos valores coinciden pero a efectos prácticos, puesto que el PK es autoincremental y en este caso el procedimiento no lo pide como parámetro, se lo asigna el Sistema Gestor de la Base de Datos).

```
--Insercion de productos CON el procedimiento spInsertarProducto
EXEC spInsertarProducto 'Laptop ASUS VivoBook 15', 1, 500.00, 650.00, 15, 5, 'no', 4, NULL;
EXEC spInsertarProducto 'Monitor ASUS 24" Full HD', 2, 150.00, 200.00, 30, 10, 'no', 4, NULL;
EXEC spInsertarProducto 'Teclado ASUS Gaming', 3, 50.00, 75.00, 40, 15, 'no', 4, NULL;
EXEC spInsertarProducto 'Ratón ASUS Gaming', 3, 30.00, 50.00, 35, 10, 'no', 4, NULL;
EXEC spInsertarProducto 'Laptop ASUS ROG Strix', 1, 1000.00, 1300.00, 20, 5, 'no', 4, NULL;
EXEC spInsertarProducto 'Auriculares ASUS ROG', 3, 70.00, 100.00, 50, 20, 'no', 4, NULL;
EXEC spInsertarProducto 'SSD ASUS 500GB', 2, 80.00, 120.00, 25, 10, 'no', 4, NULL;
EXEC spInsertarProducto 'Memoria RAM ASUS 16GB', 2, 60.00, 90.00, 40, 15, 'no', 4, NULL;
EXEC spInsertarProducto 'Laptop ASUS ZenBook', 1, 750.00, 950.00, 18, 5, 'no', 4, NULL;
EXEC spInsertarProducto 'Monitor ASUS 27" 4K', 2, 300.00, 450.00, 12, 5, 'no', 4, NULL;
EXEC spInsertarProducto 'Mousepad ASUS RGB', 3, 20.00, 35.00, 50, 20, 'no', 4, NULL;
EXEC spInsertarProducto 'Laptop ASUS TUF Gaming', 1, 800.00, 1050.00, 22, 7, 'no', 4, NULL;
EXEC spInsertarProducto 'Mouse ASUS TUF', 3, 40.00, 60.00, 38, 12, 'no', 4, NULL;
EXEC spInsertarProducto 'Teclado ASUS TUF', 3, 60.00, 85.00, 30, 12, 'no', 4, NULL;
EXEC spInsertarProducto 'Adaptador ASUS WiFi', 2, 15.00, 25.00, 60, 25, 'no', 4, NULL;
EXEC spInsertarProducto 'Laptop ASUS ExpertBook', 1, 950.00, 1250.00, 19, 6, 'no', 4, NULL;
EXEC spInsertarProducto 'Cargador ASUS Laptop', 2, 30.00, 45.00, 40, 15, 'no', 4, NULL;
EXEC spInsertarProducto 'Mochila ASUS Gaming', 3, 40.00, 60.00, 25, 10, 'no', 4, NULL;
EXEC spInsertarProducto 'Pantalla ASUS 32" Curva', 2, 400.00, 550.00, 15, 5, 'no', 4, NULL;
EXEC spInsertarProducto 'Cable HDMI ASUS', 2, 10.00, 20.00, 50, 20, 'no', 4, NULL;
EXEC spInsertarProducto 'Laptop ASUS X515', 1, 400.00, 550.00, 25, 10, 'no', 4, NULL;
EXEC spInsertarProducto 'Pantalla ASUS 24" Full HD', 2, 140.00, 180.00, 30, 12, 'no', 4, NULL;
EXEC spInsertarProducto 'Teclado ASUS Wireless', 3, 45.00, 70.00, 50, 20, 'no', 4, NULL;
EXEC spInsertarProducto 'Laptop ASUS Flip 14', 1, 650.00, 850.00, 20, 8, 'no', 4, NULL;
EXEC spInsertarProducto 'Monitor ASUS 34" Curvo', 2, 500.00, 700.00, 12, 5, 'no', 4, NULL;
EXEC spInsertarProducto 'Disco Duro ASUS 1TB', 2, 90.00, 130.00, 20, 8, 'no', 4, NULL;
EXEC spInsertarProducto 'Mochila ASUS 15"', 3, 30.00, 45.00, 35, 12, 'no', 4, NULL;
EXEC spInsertarProducto 'Auriculares ASUS 7.1', 3, 100.00, 130.00, 25, 10, 'no', 4, NULL;
EXEC spInsertarProducto 'Laptop ASUS TUF Dash', 1, 950.00, 1200.00, 20, 7, 'no', 4, NULL;
EXEC spInsertarProducto 'Teclado ASUS RGB', 3, 55.00, 80.00, 45, 15, 'no', 4, NULL;
EXEC spInsertarProducto 'SSD ASUS 1TB', 2, 120.00, 170.00, 30, 12, 'no', 4, NULL;
```

99 %

Messages

(1 row affected)
Producto insertado exitosamente

(1 row affected)
Producto insertado exitosamente

99 %

Query executed successfully. DESKTOP-BQ97A1E\SQLEXPRESS ... DESKTOP-BQ97A1E\Usuari... proyecto_bd_J 00:00:00 0 rows

Puede notarse que el tiempo de ejecución fue el mismo, debido a que el lote de datos es relativamente pequeño.

4.2.2.2. Modificar Datos en el Modelo

Con los registros agregados anteriormente, todos de la marca 'ASUS', procedemos a modificar sus precios. De forma 'manual', primero vemos el tamaño del lote ASUS, 82 registros luego de los INSERT y spInsertarProducto.

| | id_producto | descripcion | id_categoria | precio_costo | precio_venta | stock | stock_min | eliminado | marca_prod | imagen |
|----|-------------|--------------------------|--------------|--------------|--------------|-------|-----------|-----------|------------|--------|
| 1 | 3 | Placa Base ASUS ROG | 15 | 150 | 165 | 20 | 5 | no | 4 | NULL |
| 2 | 43 | Router ASUS RT-AX88U | 9 | 250 | 275 | 10 | 2 | no | 4 | NULL |
| 3 | 51 | Laptop ASUS VivoBook 15 | 1 | 500 | 650 | 15 | 5 | no | 4 | NULL |
| 4 | 52 | Monitor ASUS 24" Full HD | 2 | 150 | 200 | 30 | 10 | no | 4 | NULL |
| 5 | 53 | Teclado ASUS Gaming | 3 | 50 | 75 | 40 | 15 | no | 4 | NULL |
| 6 | 54 | Ratón ASUS Gaming | 3 | 30 | 50 | 35 | 10 | no | 4 | NULL |
| 7 | 55 | Laptop ASUS ROG Strix | 1 | 1000 | 1300 | 20 | 5 | no | 4 | NULL |
| 8 | 56 | Auriculares ASUS ROG | 3 | 70 | 100 | 50 | 20 | no | 4 | NULL |
| 9 | 57 | SSD ASUS 500GB | 2 | 80 | 120 | 25 | 10 | no | 4 | NULL |
| 10 | 58 | Memoria RAM ASUS 16... | 2 | 60 | 90 | 40 | 15 | no | 4 | NULL |
| 11 | 59 | Laptop ASUS ZenBook | 1 | 750 | 950 | 18 | 5 | no | 4 | NULL |

Ahora actualizamos los precios, afectando a los 82 registros:

```
--Modificación de Precios con instrucciones básicas SQL, un 25% de aumento
UPDATE productos
SET precio_venta = precio_venta * 1.25
WHERE marca_prod = (
    SELECT id_marca
    FROM marcas
    WHERE nombre_marca = 'ASUS'
);
--Productos afectados por la actualización
SELECT *
FROM productos
WHERE marca_prod = (
    SELECT id_marca
    FROM marcas
    WHERE nombre_marca = 'ASUS'
);
```

Verificamos que dichos cambios se realizaron:

| | id_producto | descripcion | id_categoria | precio_costo | precio_venta | stock | stock_min | eliminado | marca_prod | imagen |
|----|-------------|--------------------------|--------------|--------------|--------------|-------|-----------|-----------|------------|--------|
| 1 | 3 | Placa Base ASUS ROG | 15 | 150 | 206.25 | 20 | 5 | no | 4 | NULL |
| 2 | 43 | Router ASUS RT-AX88U | 9 | 250 | 343.75 | 10 | 2 | no | 4 | NULL |
| 3 | 51 | Laptop ASUS VivoBook 15 | 1 | 500 | 812.5 | 15 | 5 | no | 4 | NULL |
| 4 | 52 | Monitor ASUS 24" Full HD | 2 | 150 | 250 | 30 | 10 | no | 4 | NULL |
| 5 | 53 | Teclado ASUS Gaming | 3 | 50 | 93.75 | 40 | 15 | no | 4 | NULL |
| 6 | 54 | Ratón ASUS Gaming | 3 | 30 | 62.5 | 35 | 10 | no | 4 | NULL |
| 7 | 55 | Laptop ASUS ROG Strix | 1 | 1000 | 1625 | 20 | 5 | no | 4 | NULL |
| 8 | 56 | Auriculares ASUS ROG | 3 | 70 | 125 | 50 | 20 | no | 4 | NULL |
| 9 | 57 | SSD ASUS 500GB | 2 | 80 | 150 | 25 | 10 | no | 4 | NULL |
| 10 | 58 | Memoria RAM ASUS 16GB | 2 | 60 | 112.5 | 40 | 15 | no | 4 | NULL |
| 11 | 59 | Laptop ASUS ZenBook | 1 | 750 | 1187.5 | 18 | 5 | no | 4 | NULL |

Ahora, haremos el mismo aumento, pero con el tipo procedural. Vemos de entrada que la codificación resulta mucho más sencilla.

```
--Modificación de Precios con instrucciones básicas SQL, un 25% de aumento
EXECUTE spActualizarPrecioVentaMarca 'ASUS', 1.25 --aumenta el precio en un 25%
```

Y sus resultados son:

| | id_producto | descripcion | id_categoria | precio_costo | precio_venta | stock | stock_min | eliminado | marca_prod | imagen |
|---|-------------|--------------------------|--------------|--------------|--------------|-------|-----------|-----------|------------|--------|
| 1 | 3 | Placa Base ASUS ROG | 15 | 150 | 257.8125 | 20 | 5 | no | 4 | NULL |
| 2 | 43 | Router ASUS RT-AX88U | 9 | 250 | 429.6875 | 10 | 2 | no | 4 | NULL |
| 3 | 51 | Laptop ASUS VivoBook 15 | 1 | 500 | 1015.625 | 15 | 5 | no | 4 | NULL |
| 4 | 52 | Monitor ASUS 24" Full HD | 2 | 150 | 312.5 | 30 | 10 | no | 4 | NULL |
| 5 | 53 | Teclado ASUS Gaming | 3 | 50 | 117.1875 | 40 | 15 | no | 4 | NULL |

Nuevamente, el tiempo de ejecución no difiere debido al tamaño de la muestra. Pero ya resulta cómodo apreciar la diferencia de líneas de código entre un método y otro.

4.2.2.3. Borrar Datos en el Modelo

Ahora eliminamos un tipo de producto por nombre usando el procedimiento:

```
--Eliminar producto insertado
EXEC spEliminarProducto 'Ratón ASUS TUF Gaming'; --eliminando el producto que creamos
```

99 %

Messages

(2 rows affected)
Producto eliminado exitosamente
Completion time: 2024-11-11T13:54:26.2827945-03:00

99 %

Query executed successfully.

Verificando, observamos ahora 80 registros ASUS.

```
--Productos afectados por la actualización
SELECT *
FROM productos
WHERE marca_prod = (
    SELECT id_marca
    FROM marcas
    WHERE nombre_marca = 'ASUS'
);

--Eliminar producto insertado
EXEC spEliminarProducto 'Ratón ASUS TUF Gaming'; --eliminando el producto que creamos
```

99 %

Results

| | id_producto | descripcion | id_categoria | precio_costo | precio_venta | stock | stock_min | eliminado | marca_prod | imagen |
|---|-------------|--------------------------|--------------|--------------|--------------|-------|-----------|-----------|------------|--------|
| 1 | 3 | Placa Base ASUS ROG | 15 | 150 | 257.8125 | 20 | 5 | no | 4 | NULL |
| 2 | 43 | Router ASUS RT-AX88U | 9 | 250 | 429.6875 | 10 | 2 | no | 4 | NULL |
| 3 | 51 | Laptop ASUS VivoBook 15 | 1 | 500 | 1015.625 | 15 | 5 | no | 4 | NULL |
| 4 | 52 | Monitor ASUS 24" Full HD | 2 | 150 | 312.5 | 30 | 10 | no | 4 | NULL |
| 5 | 53 | Teclado ASUS Gaming | 3 | 50 | 117.1875 | 40 | 15 | no | 4 | NULL |

Query executed successfully.

DESKTOP-BQ97A1E\SQLSERVER ... | DESKTOP-BQ97A1E\Usuari... | proyecto_bd_1 | 00:00:00 | 80 rows

4.2.3. Funciones Definidas por el Usuario

Primero, una función que nos permite tomar todos los registros de una misma marca.

```
--4. Función para traer los registros de una misma marca
CREATE FUNCTION fnTotalMarca(
    @NombreMarca VARCHAR(30)
)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM productos p
    WHERE p.marca_prod IN (
        SELECT m.id_marca
        FROM marcas m
        WHERE nombre_marca = @NombreMarca
    )
);

SELECT * FROM fnTotalMarca('Intel');
```

| Results | | Messages | | | | | | | | |
|---------|-------------|--------------------------|--------------|--------------|--------------|-------|-----------|-----------|------------|--------|
| | id_producto | descripcion | id_categoria | precio_costo | precio_venta | stock | stock_min | eliminado | marca_prod | imagen |
| 1 | 1 | Procesador Intel Core i9 | 13 | 350 | 648 | 10 | 2 | no | 1 | NULL |
| 2 | 47 | Procesador Intel Core i7 | 13 | 280 | 547,2 | 10 | 2 | no | 1 | NULL |
| 3 | 52 | Producto Procedimiento | 1 | 50 | 70 | 100 | 10 | No | 1 | NULL |

Luego, una función para traer todos los empleados con un mismo perfil.

--5. Funcion que trae todos los de un mismo perfil

```
CREATE FUNCTION fnUsuariosPorPerfil (
    @DescripcionPerfil VARCHAR(20)
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        u.id_usuario,
        u.apellido,
        u.nombre,
        u.usuario,
        u.email,
        u.estado_perfil
    FROM users u
    JOIN perfiles p ON u.id_perfil = p.id_perfil
    WHERE p.descripcion = @DescripcionPerfil
);

SELECT * FROM fnUsuariosPorPerfil('Vendedor');
```

| Results | | Messages | | | | |
|---------|------------|----------|----------|-----------------|-----------------------------|---------------|
| | id_usuario | apellido | nombre | usuario | email | estado_perfil |
| 1 | 3 | López | Carlos | carloslopez | carloslopez@example.com | alta |
| 2 | 8 | Díaz | Laura | lauradiaz | lauradiaz@example.com | alta |
| 3 | 13 | Ortiz | Samuel | samuelortiz | samuelortiz@example.com | alta |
| 4 | 18 | Sánchez | Mónica | monicasanchez | monicasanchez@example.com | alta |
| 5 | 23 | López | Fernando | fernandolopez | fernandolopez@example.com | alta |
| 6 | 28 | Díaz | Lucía | luciadiaz | luciadiaz@example.com | alta |
| 7 | 33 | Ortiz | Patricia | patriciaortiz | patriciaortiz@example.com | alta |
| 8 | 38 | Sánchez | Verónica | veronicasanchez | veronicasanchez@example.com | alta |
| 9 | 43 | López | Julián | julianlopez | julianlopez@example.com | alta |
| 10 | 48 | Díaz | Camila | camiladiaz | camiladiaz@example.com | alta |

Por último, una función que nos permite textualizar una fecha, en este caso la de nacimiento de un empleado, lo cual nos ayudará a que sea más estético de mostrar.

```
--6. Pasar fecha de nacimiento del usuario a formato texto
CREATE FUNCTION fnFechaNacimientoUsuarioTexto (
    @dni_usuario BIGINT
)
RETURNS VARCHAR(50)
AS
BEGIN
    DECLARE @FechaNacimiento DATE;
    DECLARE @FechaTexto VARCHAR(50);

    -- Obtener la fecha de nacimiento del cliente según su DNI
    SELECT @FechaNacimiento = fecha_nac
    FROM users
    WHERE dni_usuario = @dni_usuario;

    -- Si la fecha de nacimiento es NULL, retornar un mensaje vacío o personalizado
    IF @FechaNacimiento IS NULL
        RETURN 'Fecha no disponible';

    -- Construir la fecha en formato de texto
    SET @FechaTexto = CONCAT(
        DAY(@FechaNacimiento), ' de ',
        DATENAME(MONTH, @FechaNacimiento), ' de ',
        YEAR(@FechaNacimiento)
    );

    RETURN @FechaTexto;
END;

SELECT dbo.fnFechaNacimientoUsuarioTexto(12345678) AS FechaNacimientoTexto;
```

| Results | | Messages | |
|---------|--|----------------------|--|
| | | FechaNacimientoTexto | |
| 1 | | 10 de April de 1985 | |

Sobre la eficiencia de un método sobre el otro, nos resulta difícil sacar grandes conclusiones debido al tamaño del lote. Pero al momento de codificar resultó más cómodo el tener un procedimiento que desarrollar toda una consulta explícitamente y repitiendo el código que ya habíamos escrito antes.

4.3. Optimización de Consultas a Través de Índices.

Referenciando a lo mostrado en el punto 2.4. Sobre Optimización de Consultas a Través de Índices, el script SQL del Tema 03 contiene el siguiente desarrollo.

4.3.1 Carga de Datos: La carga masiva de datos se realizó a través de un Script en SQLServer de ejemplo que configura inserciones masivas mediante instrucciones y simula datos para las tablas.

```
DECLARE @i INT = 11;

WHILE @i <= 1000000
BEGIN
    INSERT INTO ventas (id_usuario, id_cliente, fecha_venta, total_venta, numeroVenta, estado, id_TipoFact)
    VALUES (
        (1 + @i % 50), -- id_usuario, suponiendo que tienes 1000 usuarios en la tabla 'users'
        (SELECT TOP 1 dni_c FROM clientes ORDER BY NEWID()), -- id_cliente, suponiendo que tienes 500 clientes en la tabla 'cliente'
        DATEADD(DAY, @i % 365, '2024-01-01'), -- fecha_venta con una fecha variada
        RAND() * 50000 + 500, -- total_venta, un valor aleatorio entre 50 y 550
        @i, -- numeroVenta, un número de venta único para cada registro
        'Activo', -- estado, puedes cambiarlo según tu necesidad
        (1 + @i % 10) -- id_TipoFact, suponiendo que tienes 5 tipos de factura en 'tipo_factura'
    );

    SET @i = @i + 1;
END;

DECLARE @j INT = 1;

WHILE @j <= 1000000
BEGIN
    DECLARE @id_venta INT = (SELECT TOP 1 id_venta FROM ventas ORDER BY NEWID());
    DECLARE @id_producto INT = (SELECT TOP 1 id_producto FROM productos ORDER BY NEWID());
    DECLARE @cantidad INT = ABS(CHECKSUM(NEWID())) % 10 + 1;
    DECLARE @precio DECIMAL(10, 2) = (SELECT precio_venta FROM productos WHERE id_producto = @id_producto);
    DECLARE @sub_total DECIMAL(10, 2) = @cantidad * @precio;

    INSERT INTO venta_detalle (id_venta, id_producto, cantidad, sub_total, eliminado)
    VALUES (
        @id_venta,
        @id_producto,
        @cantidad,
        @sub_total,
        'no' -- Assuming 0 for not deleted
    );

    SET @j = @j + 1;
END;
```

4.3.2 Realizacion Consulta: Una vez generado la carga de datos, realizamos una consulta para obtener las ventas en un periodo de fecha, a que usuario pertenece (Nombre y Apellido), Teléfono, el total venta, producto, precio de venta y cantidad del mismo.

```

SET STATISTICS TIME ON
SET STATISTICS IO ON
GO
SELECT
    v.id_venta,
    v.numeroVenta,
    v.fecha_venta,
    u.nombre AS usuario_nombre,
    u.apellido AS usuario_apellido,
    u.telefono,
    v.total_venta,
    p.descripcion AS producto_descripcion,
    p.precio_venta,
    vd.cantidad
FROM
    ventas v
INNER JOIN
    users u ON v.id_usuario = u.id_usuario
INNER JOIN
    venta_detalle vd ON v.id_venta = vd.id_venta
INNER JOIN
    productos p ON vd.id_producto = p.id_producto
WHERE
    v.fecha_venta BETWEEN '2023-10-01' AND '2024-11-08'
ORDER BY
    v.fecha_venta;

```

Al observar el plan de ejecución podemos ver las Estadísticas TIME e IO generadas, muestra que se realizan 7619 lecturas lógicas, con 4497 ms de time y consume 391 ms de tiempo de CPU para recuperar información.

```

SQL Server Execution Times:
    CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
    CPU time = 46 ms,  elapsed time = 507 ms.

(26946 rows affected)
Table 'ventas'. Scan count 5, logical reads 7619, physical reads 1, page server reads 0, read-ahead reads 7414
Table 'venta_detalle'. Scan count 5, logical reads 781, physical reads 1, page server reads 0, read-ahead read
Table 'productos'. Scan count 4, logical reads 4, physical reads 1, page server reads 0, read-ahead reads 0, p
Table 'users'. Scan count 5, logical reads 4, physical reads 1, page server reads 0, read-ahead reads 0, page
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, p
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, p

SQL Server Execution Times:
    CPU time = 391 ms,  elapsed time = 4497 ms.

Completion time: 2024-11-11T14:59:27.7290424-03:00

```

4.3.3 Definición Índice Agrupado: Definir un índice agrupado sobre la columna fecha y repetir la consulta anterior. Registrar el plan de ejecución utilizado por el motor y los tiempos de respuesta.

```
CREATE NONCLUSTERED INDEX IDX_Ventas_FechaVenta ON ventas (fecha_venta);
```

Una vez creado el índice el cual lo nombramos como IDX_Ventas_FechaVenta observamos el plan de ejecución y podemos ver las Estadísticas TIME e IO generadas, muestra que se realizan 7541 lecturas lógicas, con 2866 ms de time y consume 358 ms de tiempo de CPU para recuperar información.


```

SQL Server parse and compile time:
  CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 0 ms.

SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
  CPU time = 47 ms, elapsed time = 101 ms.

(26946 rows affected)
Table 'ventas'. Scan count 5, logical reads 7541, physical reads 1, page server reads 0, read-ahead reads 7414
Table 'venta_detalle'. Scan count 5, logical reads 781, physical reads 1, page server reads 0, read-ahead read
Table 'users'. Scan count 3, logical reads 4, physical reads 1, page server reads 0, read-ahead reads 0, page
Table 'productos'. Scan count 5, logical reads 4, physical reads 1, page server reads 0, read-ahead reads 0, p
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, p
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, p

SQL Server Execution Times:
  CPU time = 358 ms,  elapsed time = 2866 ms.

Completion time: 2024-11-11T15:09:42.4729321-03:00

```

4.3.4 Borrar el índice creado

```

DROP INDEX [IDX_Ventas_FechaVenta_Incluido] ON [ventas]
GO

```

4.3.5 Definición índice agrupado incluyendo columnas: Definir otro índice agrupado sobre la columna fecha pero que además incluya las columnas seleccionadas y repetir la consulta anterior. Registrar el plan de ejecución utilizado por el motor y los tiempos de respuesta.

```

CREATE NONCLUSTERED INDEX IDX_Ventas_FechaVenta_Incluido
ON ventas (fecha_venta)
INCLUDE (numeroVenta, id_cliente, total_venta);

DROP INDEX [IDX_Ventas_FechaVenta_Incluido] ON [ventas]
GO

```

una vez creado el índice el cual lo nombramos como `IDX_Ventas_FechaVenta_Incluido`, Volvemos a ejecutar la consulta anterior utilizada y podemos observar el plan de ejecución, en lo que podemos ver las Estadísticas TIME e IO generadas, muestra que se realizan 7766 lecturas lógicas, con 470 ms de time y consume 343 ms de tiempo de CPU para recuperar información.

```

SQL Server parse and compile time:
  CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 0 ms.

SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
  CPU time = 31 ms, elapsed time = 52 ms.

(26946 rows affected)
Table 'ventas'. Scan count 5, logical reads 7766, physical reads 0, page server reads 0, read-ahead reads 16,
Table 'venta_detalle'. Scan count 5, logical reads 781, physical reads 1, page server reads 0, read-ahead re:
Table 'users'. Scan count 5, logical reads 4, physical reads 1, page server reads 0, read-ahead reads 0, page
Table 'productos'. Scan count 5, logical reads 4, physical reads 1, page server reads 0, read-ahead reads 0,
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0,
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0,

SQL Server Execution Times:
  CPU time = 343 ms,  elapsed time = 470 ms.

Completion time: 2024-11-11T15:17:06.6498517-03:00

```


4.4. Triggers.

4.4.1 Triggers de auditoría para UPDATE y DELETE

4.4.1.1 Creamos una tabla auxiliar para registrar los cambios de auditoría

```
CREATE TABLE [dbo].[auditoria] (  
    [id_auditoria] INT PRIMARY KEY IDENTITY(1,1),  
    [tabla] VARCHAR(50) NOT NULL,  
    [tipo_operacion] VARCHAR(10) NOT NULL, -- 'UPDATE' o 'DELETE'  
    [datos_anteriores] NVARCHAR(MAX) NOT NULL,  
    [fecha_operacion] DATETIME DEFAULT GETDATE(),  
    [usuario] VARCHAR(50) DEFAULT SYSTEM_USER  
);
```

4.4.1.2 Trigger para UPDATE. Este trigger registra los valores antes de la modificación

```
CREATE TRIGGER trg_auditoria_update  
ON [dbo].[clientes] -- Cambia 'clientes' por cualquier otra tabla donde se quiera aplicar auditoría  
AFTER UPDATE  
AS  
BEGIN  
    SET NOCOUNT ON;  
  
    INSERT INTO auditoria (tabla, tipo_operacion, datos_anteriores, fecha_operacion, usuario)  
    SELECT  
        'clientes' AS tabla,  
        'UPDATE' AS tipo_operacion,  
        CONCAT(  
            'dni_c:', CAST(d.dni_c AS NVARCHAR), ', ',  
            'nombre_c:', d.nombre_c, ', ',  
            'apellido:', d.apellido, ', ',  
            'fechaNac_c:', CAST(d.fechaNac_c AS NVARCHAR), ', ',  
            'email:', d.email, ', ',  
            'direccion:', d.direccion, ', ',  
            'telefono:', CAST(d.telefono AS NVARCHAR), ', ',  
            'eliminado:', d.eliminado  
        ) AS datos_anteriores,  
        GETDATE() AS fecha_operacion,  
        SYSTEM_USER AS usuario  
    FROM deleted d;  
END;  
GO
```

4.4.1.3 Trigger para DELETE. Este trigger registra los valores antes de eliminar el registro.

```
CREATE TRIGGER trg_auditoria_delete
ON [dbo].[clientes] -- Cambia 'clientes' por cualquier otra tabla donde se quiera aplicar auditoría
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO auditoria (tabla, tipo_operacion, datos_anteriores, fecha_operacion, usuario)
    SELECT
        'clientes' AS tabla,
        'DELETE' AS tipo_operacion,
        CONCAT(
            'dni_c:', CAST(d.dni_c AS NVARCHAR), ', ',
            'nombre_c:', d.nombre_c, ', ',
            'apellido:', d.apellido, ', ',
            'fechaNac_c:', CAST(d.fechaNac_c AS NVARCHAR), ', ',
            'email:', d.email, ', ',
            'direccion:', d.direccion, ', ',
            'telefono:', CAST(d.telefono AS NVARCHAR), ', ',
            'eliminado:', d.eliminado
        ) AS datos_anteriores,
        GETDATE() AS fecha_operacion,
        SYSTEM_USER AS usuario
    FROM deleted d;
END;
GO
```

4.4.2 Trigger para prevenir DELETE con mensaje personalizado. Este trigger previene las eliminaciones en una tabla específica y muestra un mensaje de error

```
CREATE TRIGGER trg_prevent_delete
ON [dbo].[clientes] -- Cambia 'clientes' por la tabla donde no se permita DELETE
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;
    RAISERROR ('No está permitido eliminar registros de la tabla clientes.', 16, 1);
    ROLLBACK TRANSACTION;
END;
GO
```

4.4.3 Pruebas para el Trigger de Auditoría (UPDATE y DELETE)

4.4.3.1 Prueba del Trigger de Auditoría en UPDATE

4.4.3.1.1 Se actualiza uno de los registros para que luego se verifique que los datos previos al cambio se registraron en la tabla de auditoría

```
1  UPDATE clientes
2  set nombre_c = 'Laurella Olmos', telefono = 1684621548
3  where nombre_c = 'Laurella';
```

Messages

```
9:07:43 PM      Started executing query at Line 1
                (1 row affected)
                Total execution time: 00:00:00.013
```

4.4.3.1.2 Se consulta la tabla de auditoría para validar el cambio del registro

```
1 SELECT * FROM [dbo].[auditoria]
2 WHERE tipo_operacion = 'UPDATE';
3
```

| Results | | Messages | |
|---------|--------------|----------|----------------|
| | id_auditoria | tabla | tipo_operacion |
| 1 | 1 | clientes | UPDATE |

| datos_anteriores | fecha_operacion | usuario |
|--|-------------------------|-------------------|
| dni_c:20048058, nombre_c:Laurella, apellido:Mantione, fec... | 2024-11-11 21:07:43.847 | RGB-REAKTOR\kille |

Como se puede apreciar, el update fue registrado en la tabla de auditoría con todos los datos

4.4.3.2 Prueba del Trigger de Auditoría en DELETE

4.4.3.2.1 Primero se desactiva el trigger de prevención de eliminación para realizar las pruebas

```
1 DISABLE TRIGGER trg_prevent_delete ON [dbo].[clientes];
2
```

Messages

```
9:22:19 PM Started executing query at Line 1
Commands completed successfully.
Total execution time: 00:00:00.005
```

4.4.3.2.2 Se elimina un registro para verificar que los datos previos al borrado se registren en la tabla de auditoría

```
1 DELETE from clientes
2 WHERE nombre_c = 'Laurella Olmos'
```

Messages

```
9:23:57 PM Started executing query at Line 1
(1 row affected)
Total execution time: 00:00:00.013
```

4.4.3.2.3 Consulta la tabla auditoria para validar el registro

```
1 SELECT * FROM [dbo].[auditoria]
2 WHERE tipo_operacion = 'DELETE'
3
```

Results Messages

| | id_auditoria | tabla | tipo_operacion |
|---|--------------|----------|----------------|
| 1 | 2 | clientes | DELETE |

| datos_anteriores | fecha_operacion | usuario |
|--|-------------------------|-------------------|
| dni_c:20048058, nombre_c:Laurella Olmos, apellido:Mantion... | 2024-11-11 21:23:57.420 | RGB-REAKTOR\kille |

Efectivamente los cambios fueron registrados con precisión en la tabla de auditoría

4.4.4 Prueba del Trigger de prevencion de DELETE

4.4.4.1 Luego de ejecutar la operación de prueba anterior volvemos a habilitar el trigger para restaurar su funcionalidad

```
1 ENABLE TRIGGER trg_prevent_delete ON [dbo].[clientes];
2
```

Messages

10:06:04 PM Started executing query at Line 1
Commands completed successfully.
Total execution time: 00:00:00.004

4.4.4.2 Ahora al intentar realizar una eliminación de la tabla se puede apreciar el mensaje “No está permitido eliminar registros de la tabla clientes” seguido de una terminación de la transacción

```
1 DELETE from clientes
2 WHERE nombre_c = 'Laurella Olmos'
```

Messages

10:09:10 PM Started executing query at Line 1
Msg 50000, Level 16, State 1, Procedure trg_prevent_delete, Line 7
No está permitido eliminar registros de la tabla clientes.
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
Total execution time: 00:00:00.007

V. CONCLUSIONES

Al desarrollar el tema de permisos y roles en SQL Server, entendimos cómo una adecuada gestión de accesos contribuye a la seguridad y eficiencia en el manejo de datos en entornos colaborativos. Desde los permisos individuales hasta la creación de roles específicos, como el de solo lectura, comprobamos que SQL Server permite personalizar y controlar el acceso a los datos de manera flexible y organizada.

En los ejercicios prácticos, implementamos permisos a nivel de usuario y de rol, asignando permisos de ejecución a un usuario con acceso limitado y comprobando cómo los roles facilitan la administración de múltiples usuarios. Esto no solo demostró que los roles permiten ahorrar tiempo y reducir errores al agregar o remover usuarios sin afectar los permisos individuales, sino que también minimiza errores administrativos, proporcionando un entorno más seguro y escalable para la administración de usuarios en la base de datos.

Hemos visto que podemos ahorrarnos líneas de código y reutilizar instrucciones mediante el uso de los procedimientos que trae el sistema por defecto y la definición de propios procesos por el usuario. Esto también ayuda a la seguridad e integridad de los datos, así como a la eficiencia en las consultas en la base de datos. Conjunto a esto, aparecen las funciones como otra alternativa, pero sin devolución de parámetros de salida y con un modo de invocación exclusivamente dentro de una consulta SELECT. Además, la posibilidad de utilizar procedimientos uno dentro del otro y vincular funciones a diferentes sesiones y usuarios según sean globales o locales.

Con las pruebas realizadas e información obtenida, pudimos observar que, al implementar índices agrupados, podemos tener mayor tiempo de respuestas y ejecución del motor de base de datos, mejorando el rendimiento de las consultas y reduciendo el tiempo de búsqueda de las mismas al contener un gran volumen de datos o bases de datos extremadamente grandes. Teniendo en cuenta que la selección de un tipo de índices incorrecto o la aplicación inadecuada, puede tener un impacto negativo en el rendimiento de las bases de datos.

Tras realizar las pruebas correspondientes, podemos concluir que los triggers implementados cumplen eficazmente su propósito, asegurando la trazabilidad y la integridad de los datos en nuestra base de datos. Los triggers de auditoría registran de manera precisa los valores previos a cualquier modificación o eliminación, incluyendo detalles como el tipo de operación, los datos afectados, la fecha, la hora y el usuario que ejecutó la acción, lo cual es fundamental para garantizar un registro detallado y confiable de las operaciones realizadas. Asimismo, el trigger que previene eliminaciones no autorizadas funcionó correctamente, bloqueando cualquier intento de borrado en la tabla protegida y mostrando el mensaje de error definido, reafirmando la seguridad de nuestras tablas más críticas. En caso de necesitar realizar operaciones excepcionales, como un DELETE, hemos comprobado que es posible deshabilitar temporalmente los triggers, lo que nos brinda la flexibilidad necesaria para gestionar escenarios imprevistos. Estas implementaciones fortalecen significativamente la seguridad y el control en nuestra base de datos, permitiéndonos trabajar con mayor confianza en la protección y auditoría de la información sensible.

VI. BIBLIOGRAFÍA.

- [¿Cuáles son los diferentes tipos de índices en SQL Server y cuándo usar cada uno de ellos? | Estrada Web Group](#)
- [Unidad 12. Triggers, procedimientos y funciones en MySQL - Apuntes de BD para DAW, DAM y ASIR -José Juan Sánchez Hernández - Curso 2023/2024](#)
- Pulido Romero, E. Escobar Domínguez, Ó. & Núñez Pérez, J. Á. (2019). Base de datos. Grupo Editorial Patria.
- Silberschatz, Korth, Sudarshan. FUNDAMENTOS DE BASES DE DATOS. Cuarta edición.
- [Qué es y cómo usar un trigger en SQL | Alura Cursos Online](#)
- [Seguimiento y optimización de consultas utilizando índices SQL Server](#)