

## 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](#_Toc182334347)

Introducción .................................................................................................................3 Definición del problema................................................................................................3 Objetivo del Trabajo Práctico........................................................................................3 Objetivos Generales.....................................................................................................3 Objetivos Específicos.................................................................................................. 4

[CAPÍTULO II: MARCO CONCEPTUAL 5](#_Toc182334348)

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](#_Toc182334349)

Descripción de cómo se realizó el Trabajo Práctico...................................................15 Herramientas……………………………………………………......................................15

[CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN DE RESULTADOS 17](#_Toc182334350)

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](#_Toc182334351)

[VI. BIBLIOGRAFÍA. 38](#_Toc182334352)

# **CAPÍTULO I: INTRODUCCIÓN**

* 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. **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. **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. **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. **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.

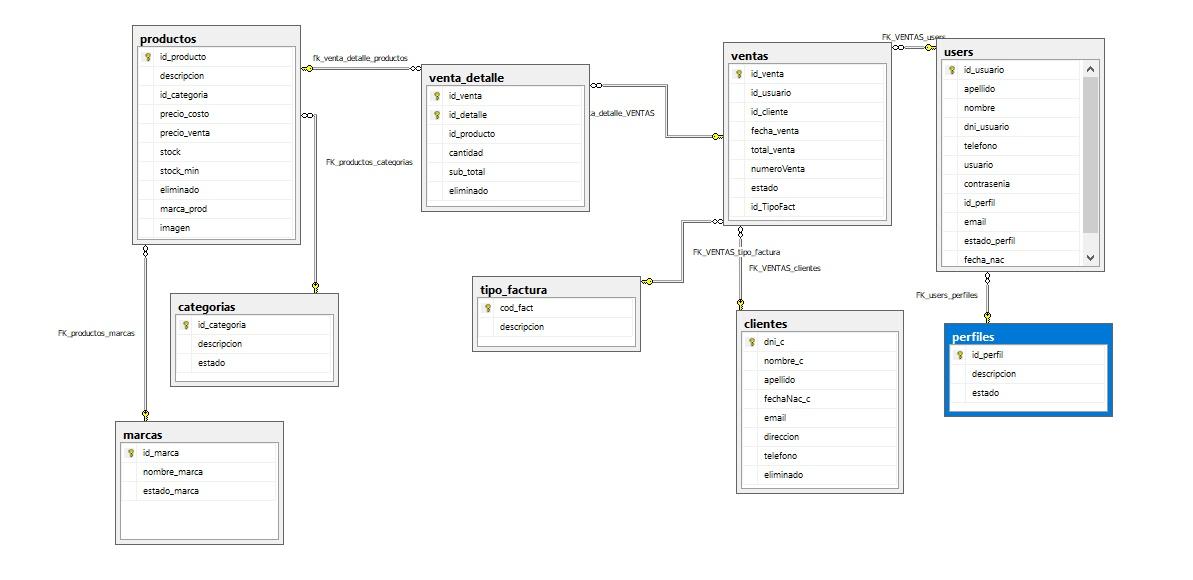
Consultas SQL para testear el funcionamiento correcto y prevenir fallas o inconsistencias dentro del sistema.

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:

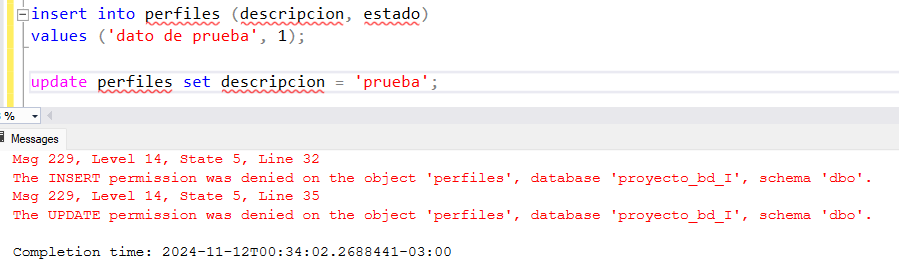
* *GRANT*: Otorga un permiso a un usuario o rol.
* *DENY*: Niega un permiso específico a un usuario o rol, incluso si se le ha otorgado anteriormente.
* *REVOKE*: Elimina un permiso previamente otorgado o denegado.

*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

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 llamó:

* Los parámetros de entrada permiten a quien realiza la llama 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**](https://estradawebgroup.com/Post/Como-definir-un-indice-agrupado-en-una-tabla-SQL-Server-con-CREATE-CLUSTERED-INDEX/20406)**:** 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**](https://estradawebgroup.com/Post/Que-son-los-indices-no-agrupados-en-SQL-Server/20650)**:** 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**](https://estradawebgroup.com/Post/Como-crear-los-indices-unicos-de-SQL-Server-con-CREATE-UNIQUE-INDEX-/20408)**:** 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](https://estradawebgroup.com/Post/Como-crear-indices-en-SQL-Server-con-la-instruccion-CREATE-INDEX/20397) en la columna **id\_producto** de la tabla productos, lo que significa que no puede haber dos productos con el mismo código.

* [**Filtered Index**](https://estradawebgroup.com/Post/Como-utilizar-Filtered-Index-para-optimizar-la-busqueda-en-grandes-bases-de-datos-en-SQL-Server/20654)**:** 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**](https://estradawebgroup.com/Post/Como-hacer-busquedas-de-texto-completo-en-SQL-Server-con-Full-Text-Index/20660)**:** 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 índex 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

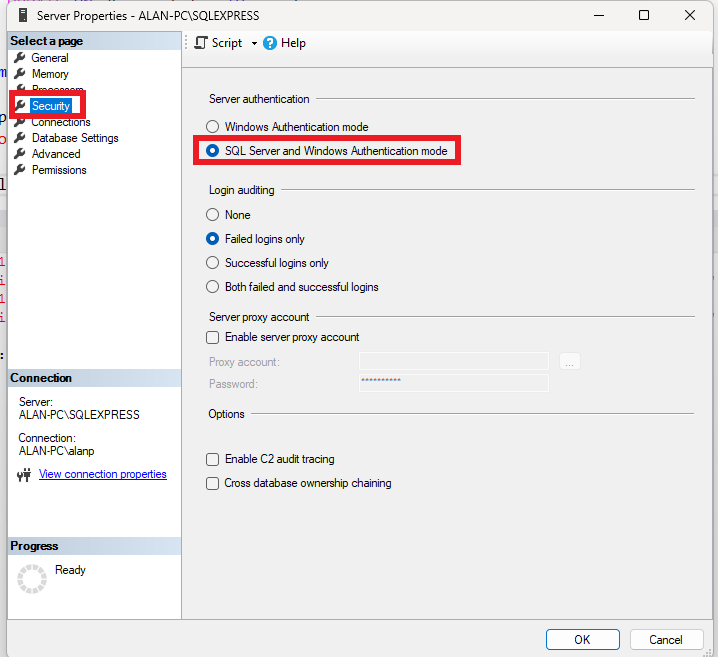
documentación del informe.

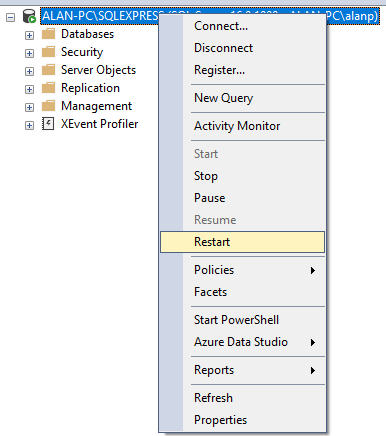
# **CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN DE RESULTADOS**

**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:

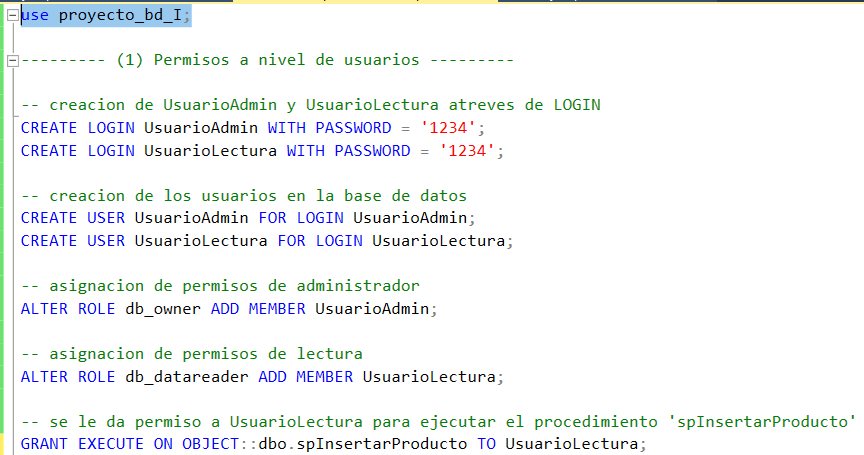
Clic derecho sobre el nombre del servidor (en mi caso, ALAN-PC\SQLEXPRESS), y seleccionar “Propiedades”.

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.

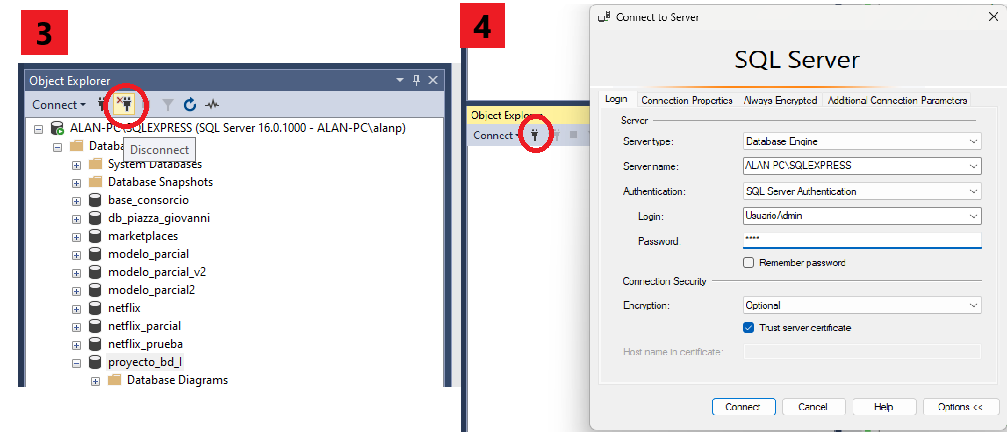
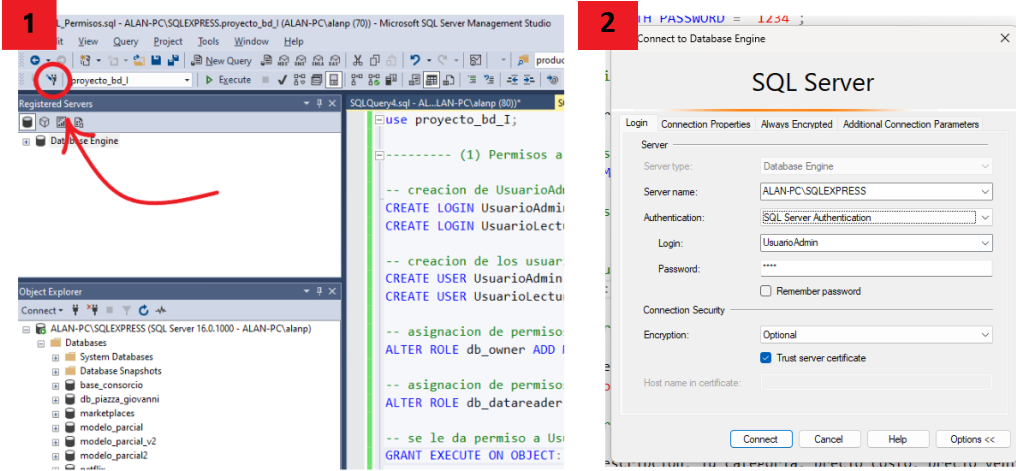
Por último debemos reiniciar el servidor de SQL Server, esto lo hacemos dándole clic derecho al nombre del servidor y seleccionando “Reiniciar”.

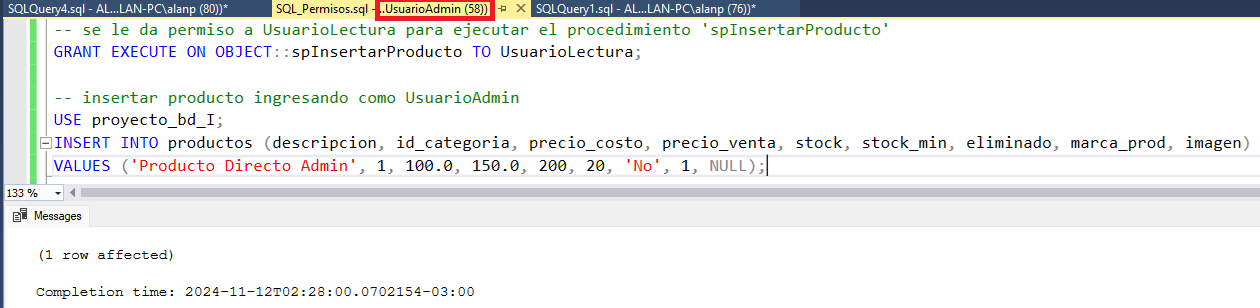
**4.1.1. Prueba de Manejo de Permisos a Nivel de Roles**

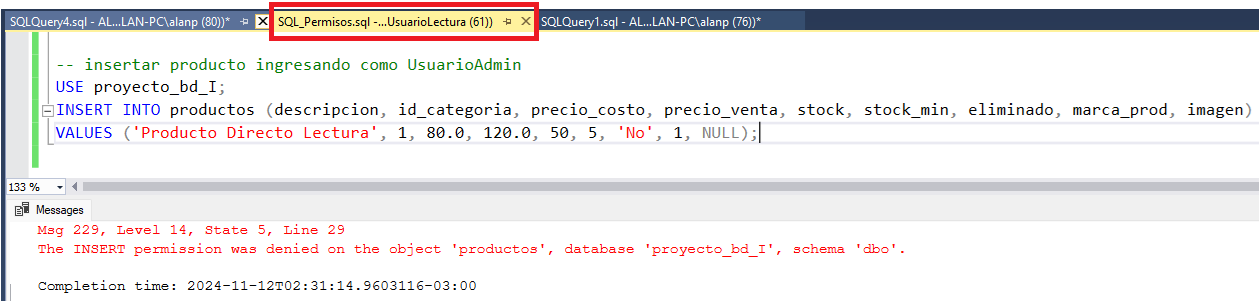
Primero se crean un par de usuarios para realizar las pruebas correspondientes, luego se le asignan los roles y permisos específicos para cada uno de ellos.

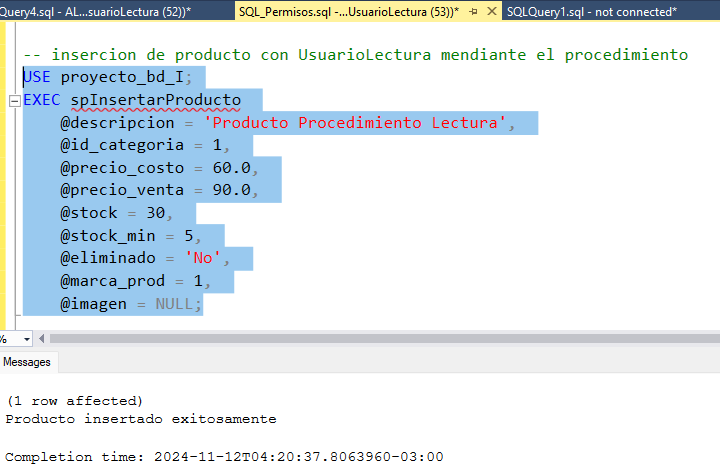


A continuación nos conectamos como UsuarioAdmin para realizar la inserción de datos.

Luego procedemos a insertar.

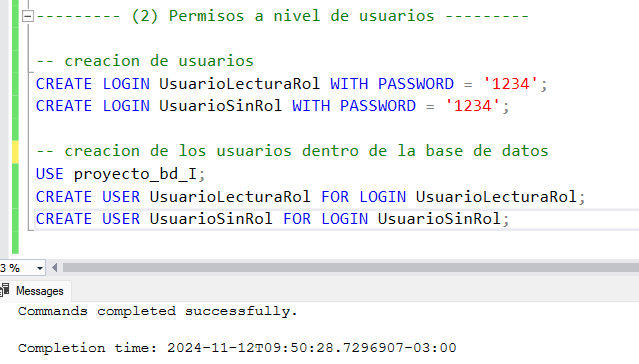
Repetimos el proceso pero conectándonos como “UsuarioLectura”.  


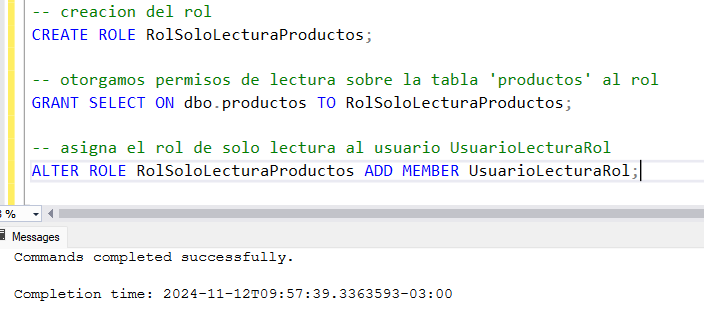
Como podemos ver las restricciones que se aplicaron sobre “UsuarioLectura” no le permite ingresar productos.

Ahora probamos insertar un producto a través del procedimiento.

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

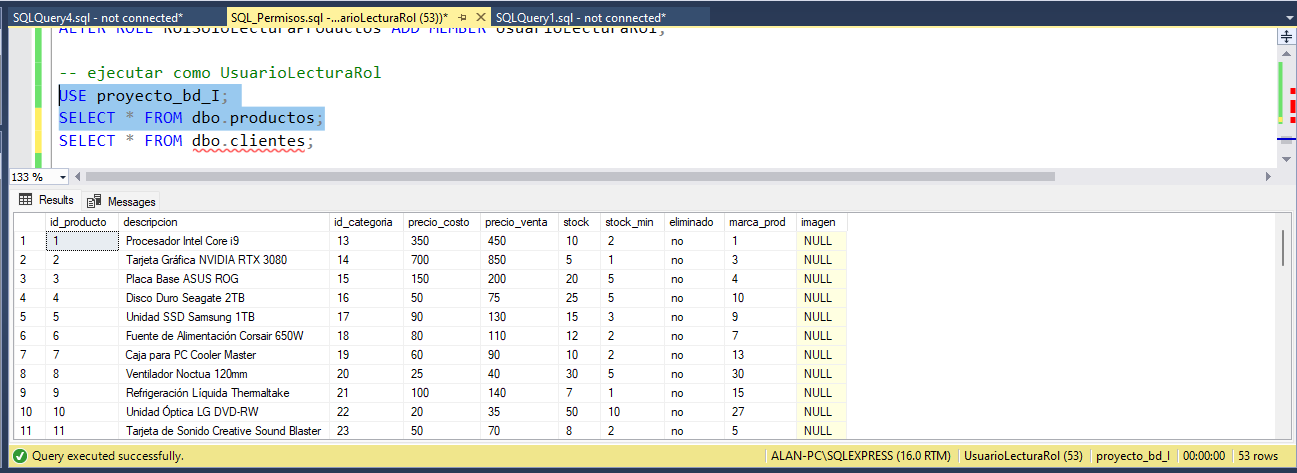
**4.1.2. Prueba de Manejo de Permisos a nivel de roles del DBMS**

Para realizar esta prueba crearemos 2 nuevos usuarios.

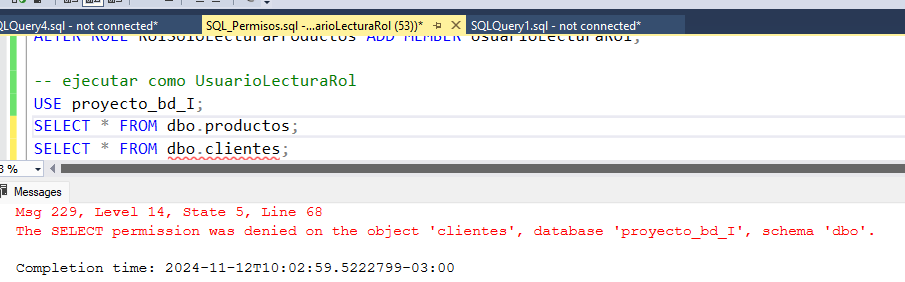
Creamos un rol que permita leer solamente la tabla de clientes, y se lo asignamos a “UsuarioLecturaRol”.

A continuación verificaremos el comportamiento de ambos usuarios.

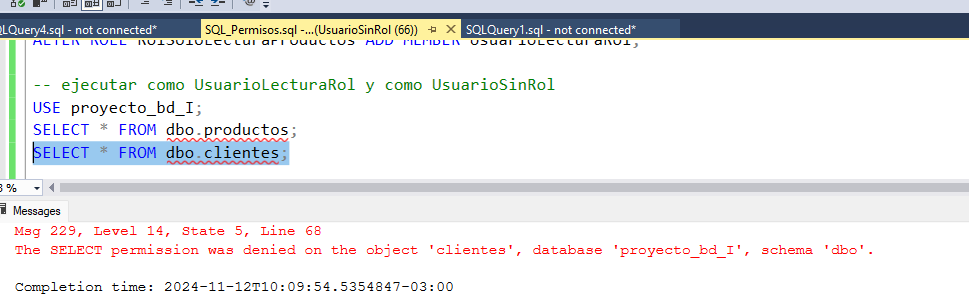
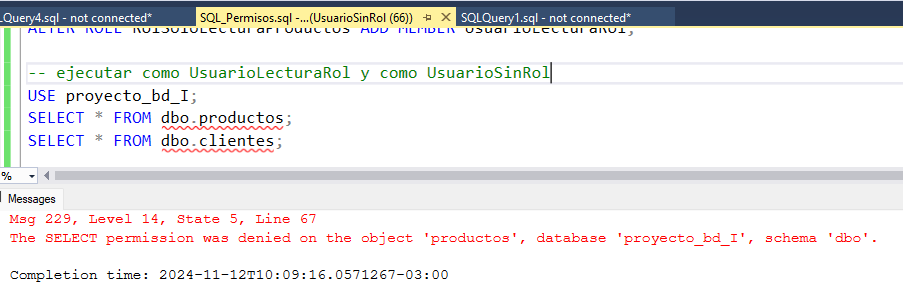
Primero nos conectamos como “UsuarioLecturaRol” y probamos leer algunas tablas, entre ellas la de productos.



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



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



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.

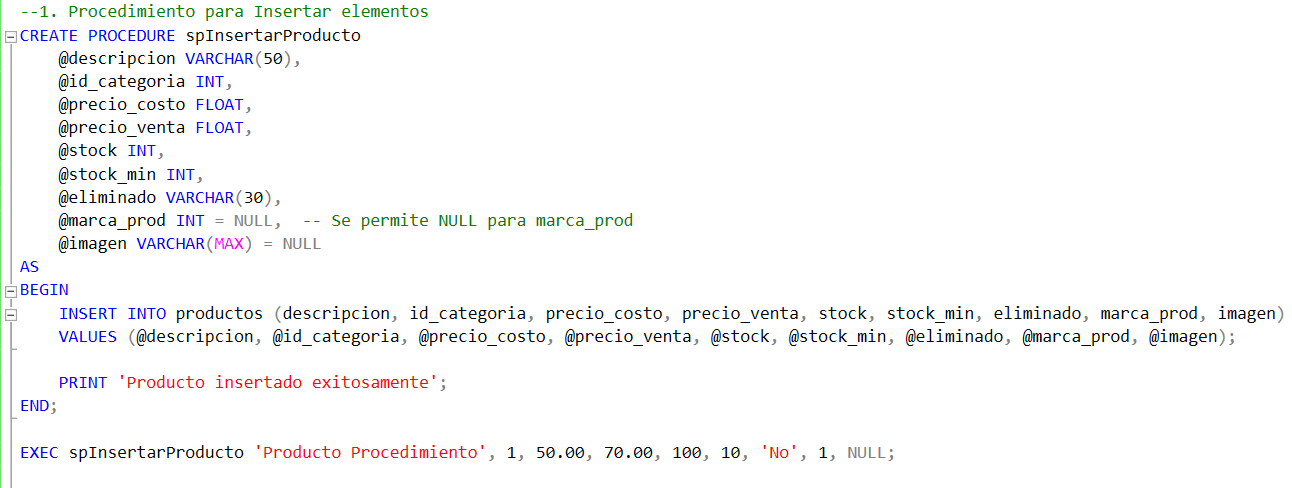
**4.2.1. Procedimientos Almacenados del Sistema**

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

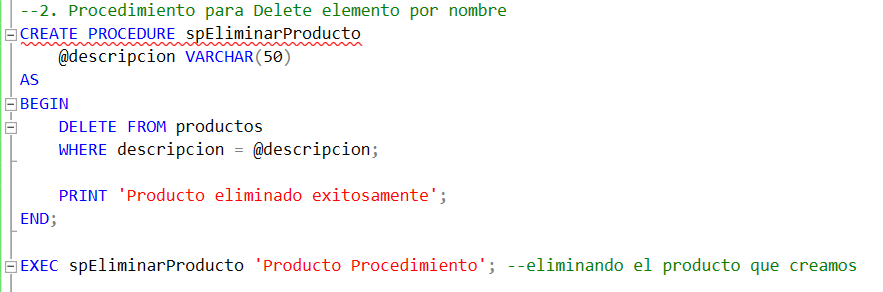


**4.2.2. Procedimientos Definidos por el Usuario**

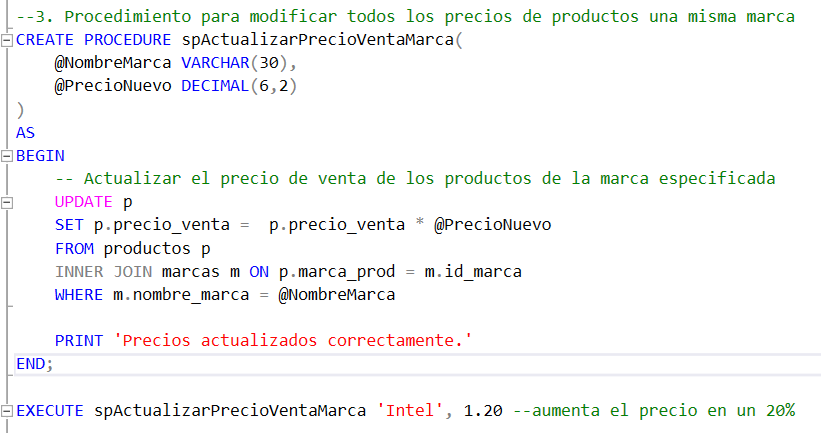
Primeramente, una función equivalente al INSERT, en este caso para añadir registros a la tabla productos.



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

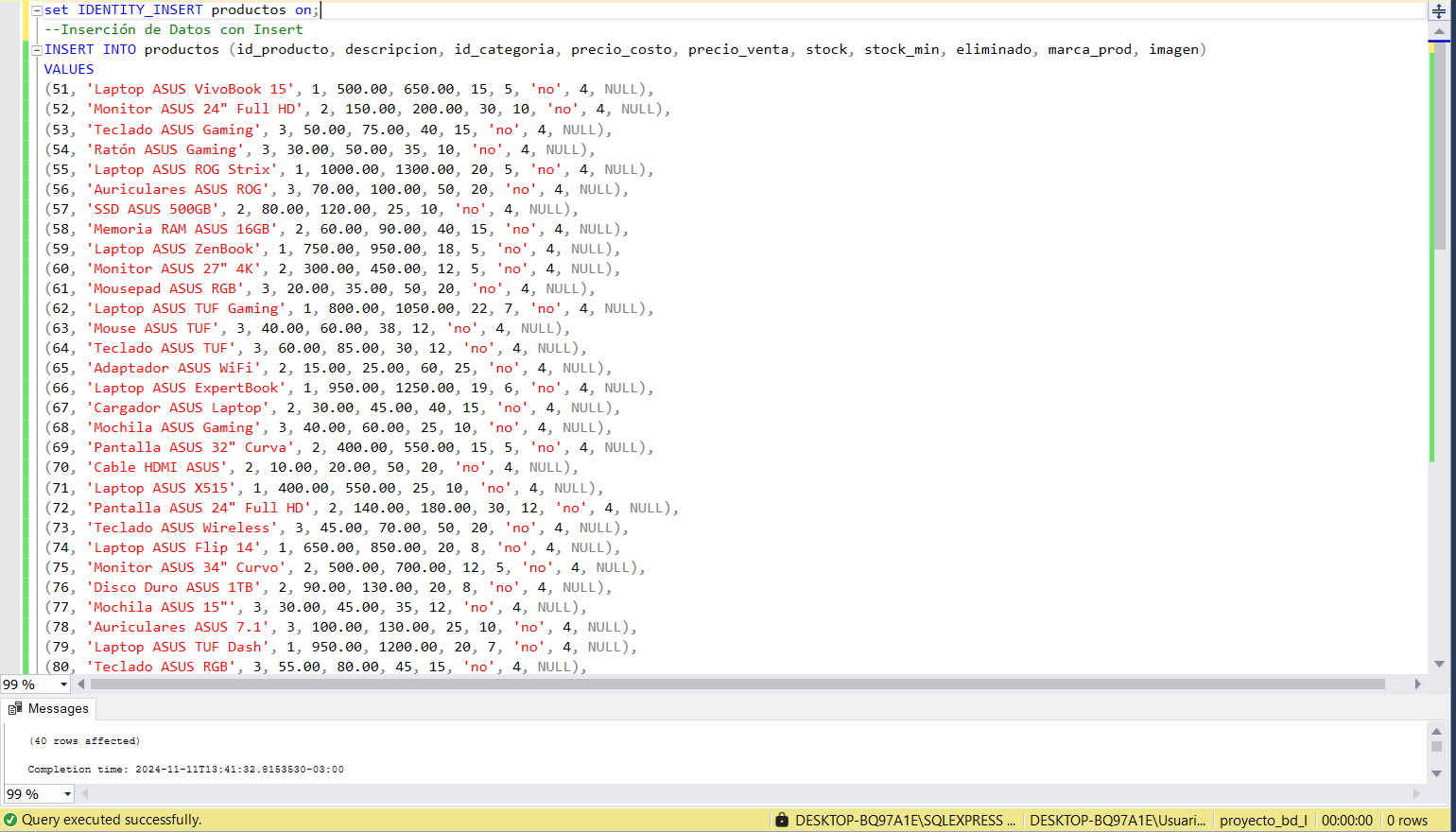


Y un procedimiento que nos será de utilidad, permitiendo alterar en igual grado los precios de una misma marca de productos.

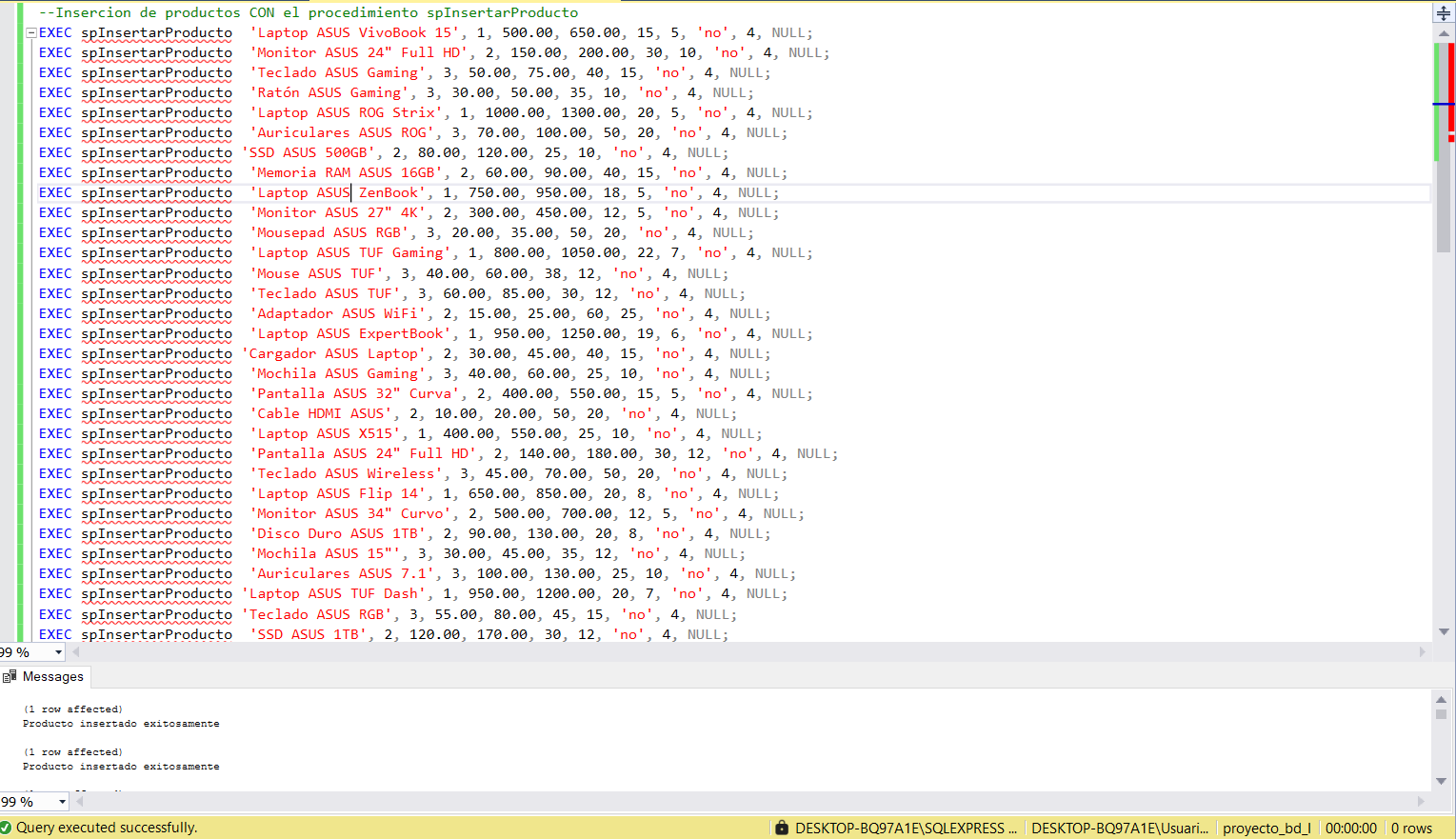


**4.2.2.1. Insertar Datos en el Modelo**

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



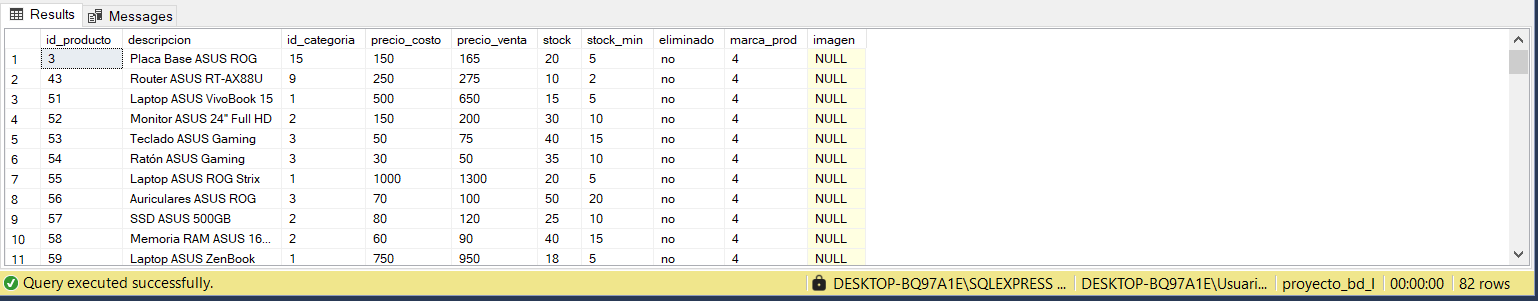
Luego, se insertaron más elementos utilizando el procedimiento creado (cuyos valores coinciden pero a efectos prácticos, puesto que el PK es autoincremental y en este caso el procedimiento no lo pide como parámetro, se lo asigna el Sistema Gestor de la Base de Datos).



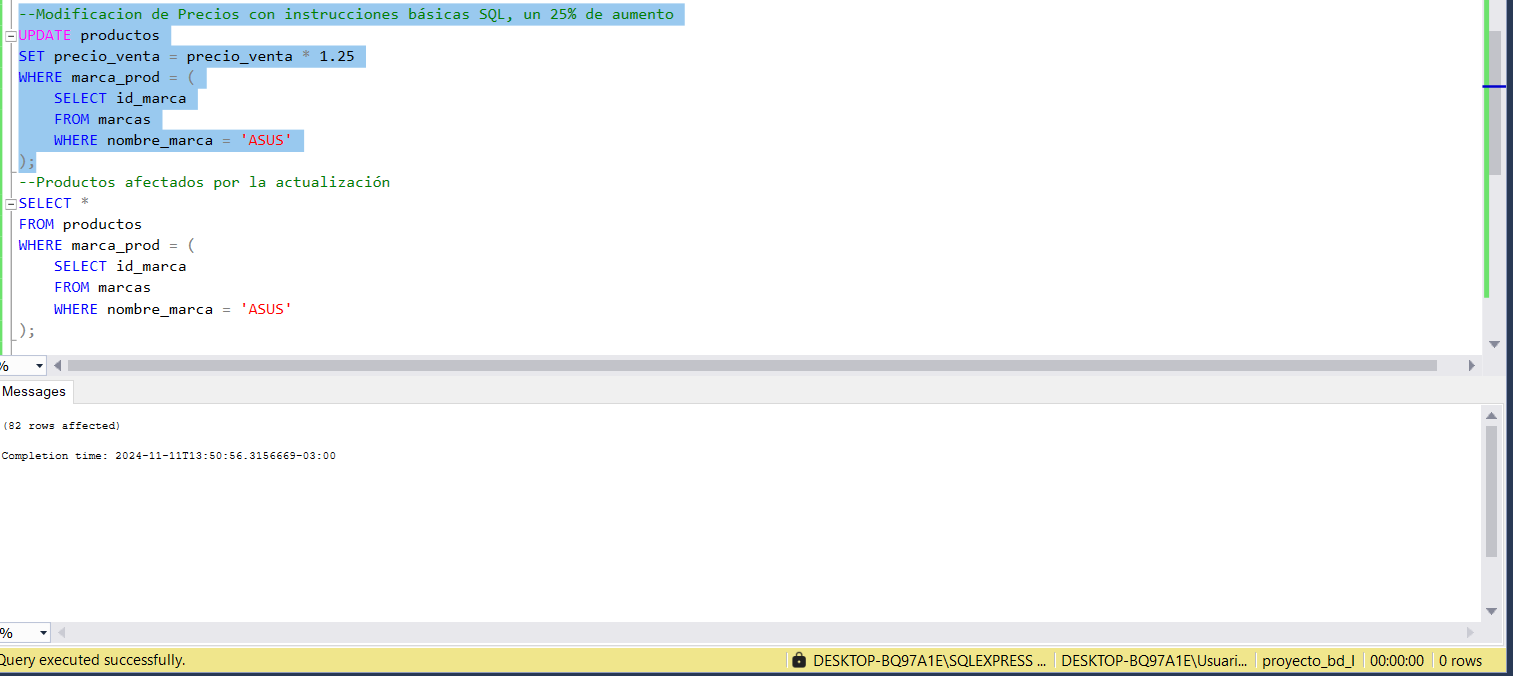
Puede notarse que el tiempo de ejecución fue el mismo, debido a que el lote de datos es relativamente pequeño.

**4.2.2.2. Modificar Datos en el Modelo**

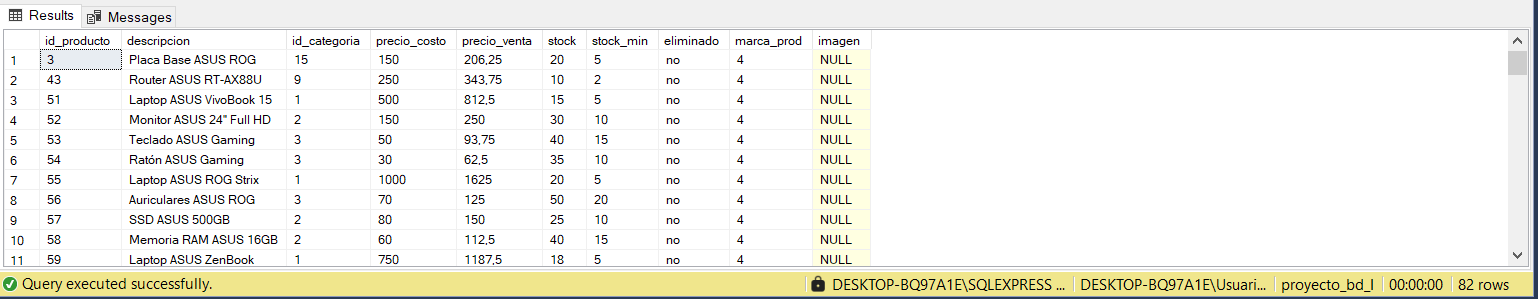
Con los registros agregados anteriormente, todos de la marca ‘ASUS’, procedemos a modificar sus precios. De forma ‘manual’, primero vemos el tamaño del lote ASUS, 82 registros luego de los INSERT y spInsertarProducto.



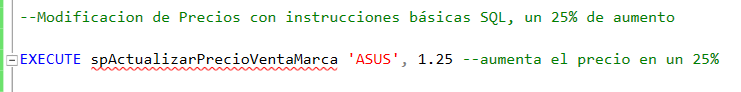
Ahora actualizamos los precios, afectando a los 82 registros:



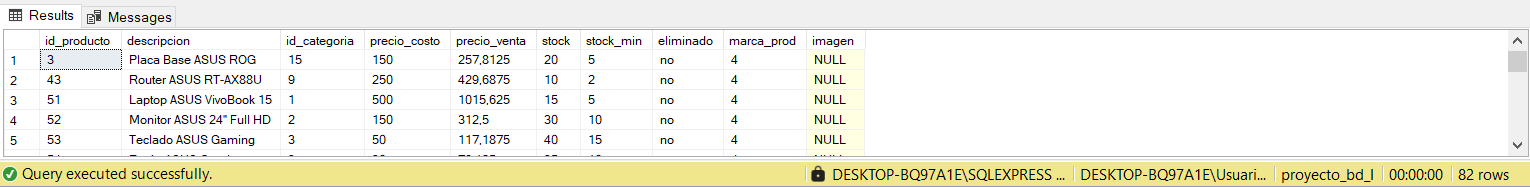
Verificamos que dichos cambios se realizaron:



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



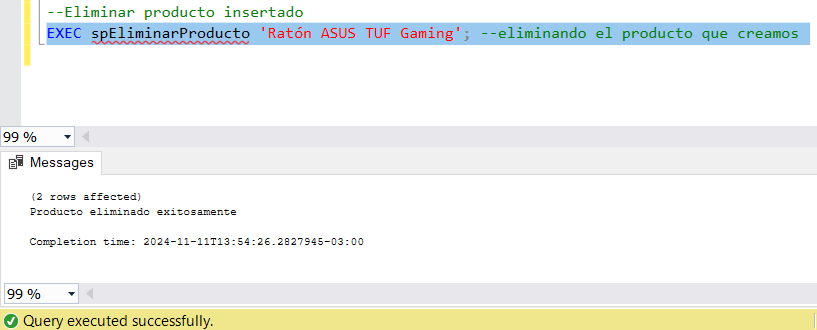
Y sus resultados son:



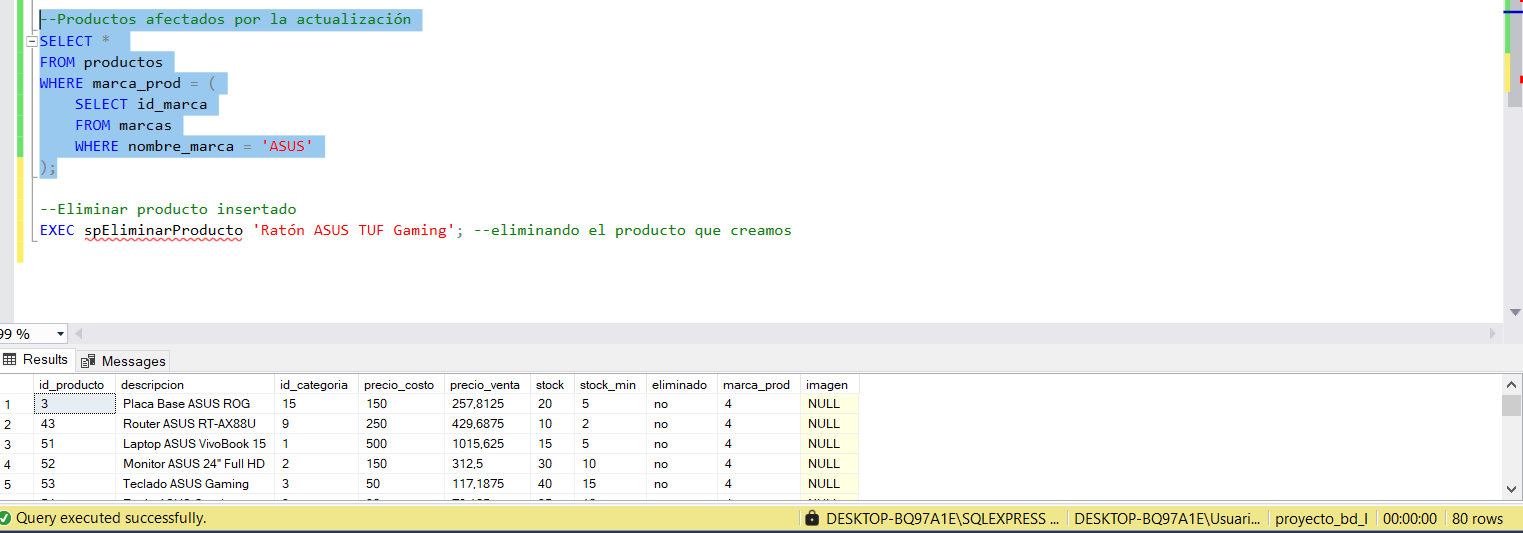
Nuevamente, el tiempo de ejecución no difiere debido al tamaño de la muestra. Pero ya resulta cómodo apreciar la diferencia de líneas de código entre un método y otro.

**4.2.2.3. Borrar Datos en el Modelo**

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

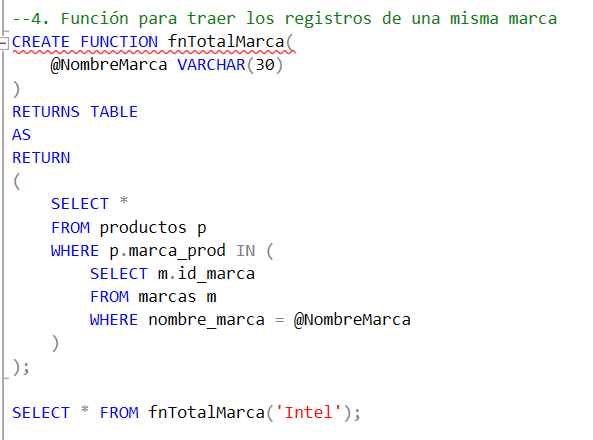


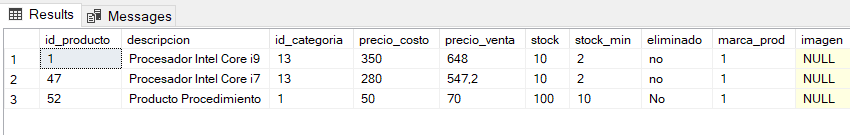
Verificando, observamos ahora 80 registros ASUS.



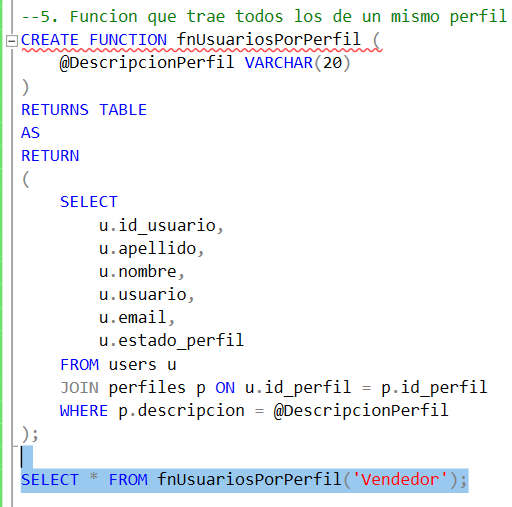
**4.2.3. Funciones Definidas por el Usuario**

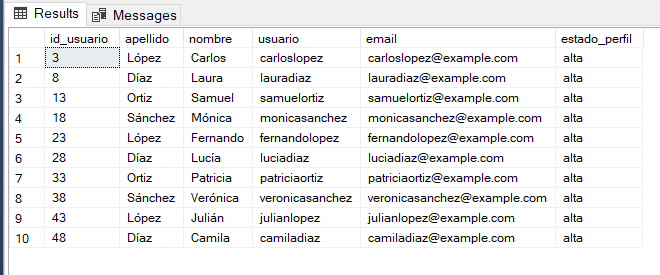
Primero, una función que nos permite tomar todos los registros de una misma marca.



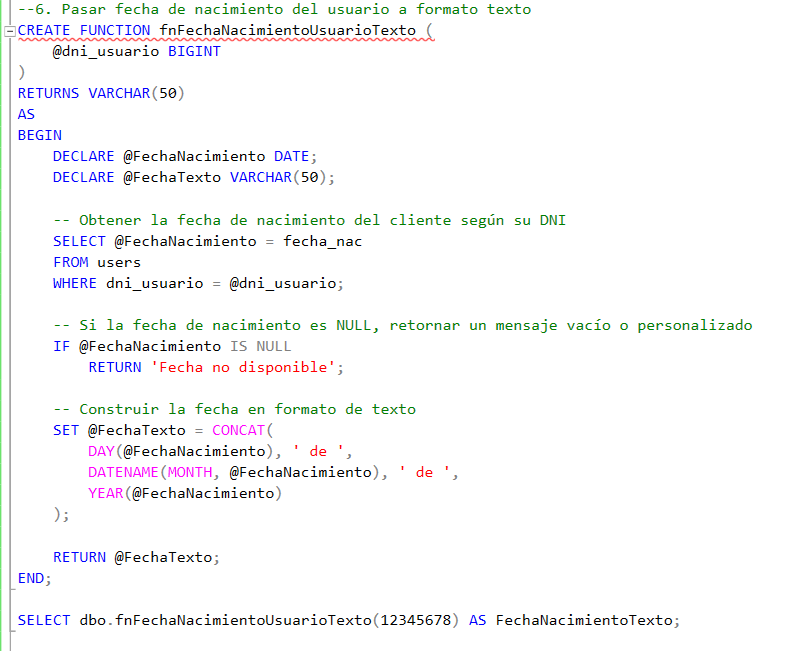


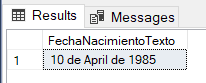
Luego, una función para traer todos los empleados con un mismo perfil.





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.



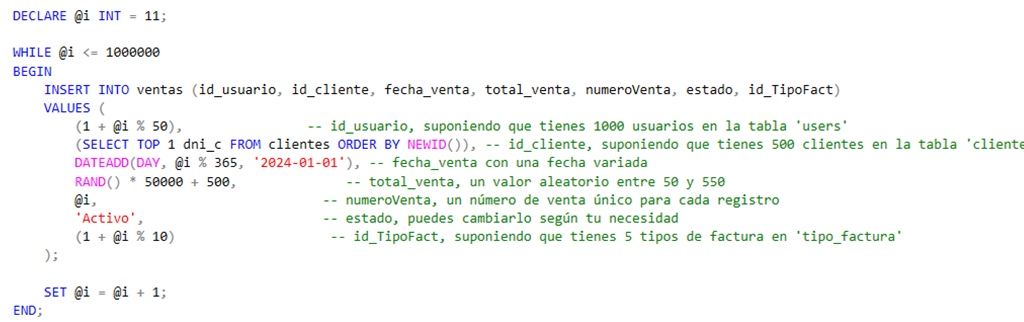


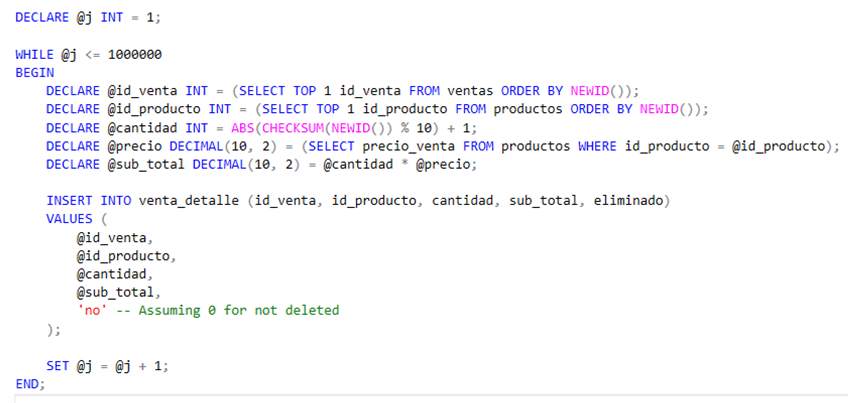
Sobre la eficiencia de un método sobre el otro, nos resulta difícil sacar grandes conclusiones debido al tamaño del lote. Pero al momento de codificar resultó más cómodo el tener un procedimiento que desarrollar toda una consulta explícitamente y repitiendo el código que ya habíamos escrito antes.

**4.3. Optimización de Consultas a Través de Índices.**

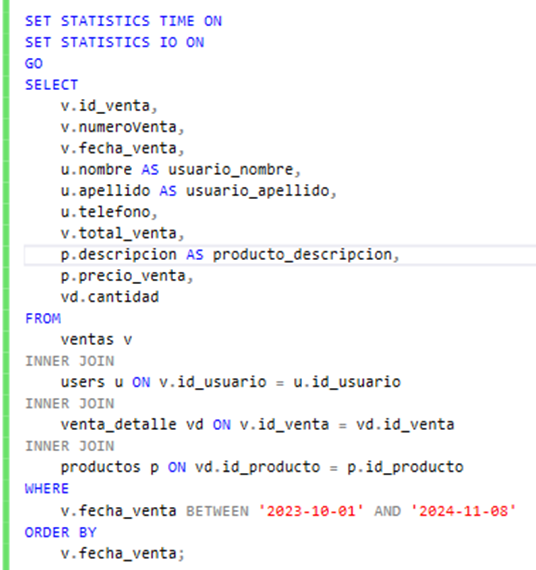
Referenciando a lo mostrado en el punto 2.4. Sobre Optimización de Consultas a Través de Índices, el script SQL del Tema 03 contiene el siguiente desarrollo.

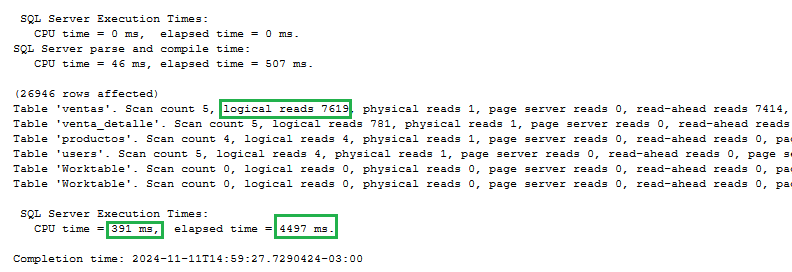
**4.3.1** Carga de Datos: La carga masiva de datos se realizó a través de un Script en SQLServer de ejemplo que configura inserciones masivas mediante instrucciones y simula datos para las tablas.





**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.

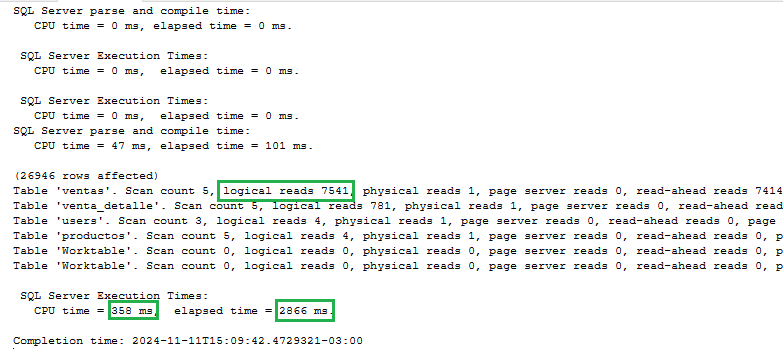
Al observar el plan de ejecución podemos ver las Estadísticas TIME e IO generadas, muestra que se realizan 7619 lecturas lógicas, con 4497 ms de time y consume 391 ms de tiempo de CPU para recuperar información.



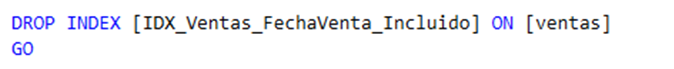
**4.3.3** **Definición Índice Agrupado:** Definir un índice agrupado sobre la columna fecha y repetir la consulta anterior. Registrar el plan de ejecución utilizado por el motor y los tiempos de respuesta.



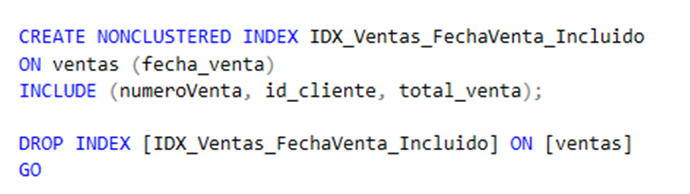
Una vez creado el índice el cual lo nombramos como IDX\_Ventas\_FechaVenta observamos el plan de ejecución y podemos ver las Estadísticas TIME e IO generadas,muestra que se realizan 7541 lecturas lógicas, con 2866 ms de time y consume 358 ms de tiempo de CPU para recuperar información.



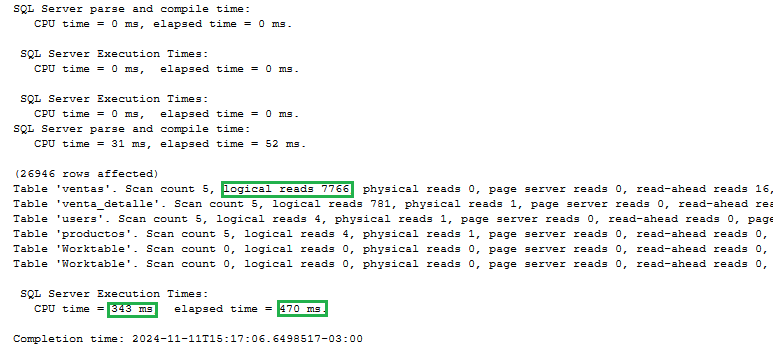
**4.3.4 Borrar el índice creado**



**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.



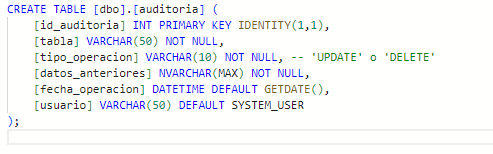
una vez creado el índice el cual lo nombramos como IDX\_Ventas\_FechaVenta\_Incluido, Volvemos a ejecutar la consulta anterior utilizada y podemos observar el plan de ejecución, en lo que podemos ver las Estadísticas TIME e IO generadas,muestra que se realizan 7766 lecturas lógicas, con 470 ms de time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



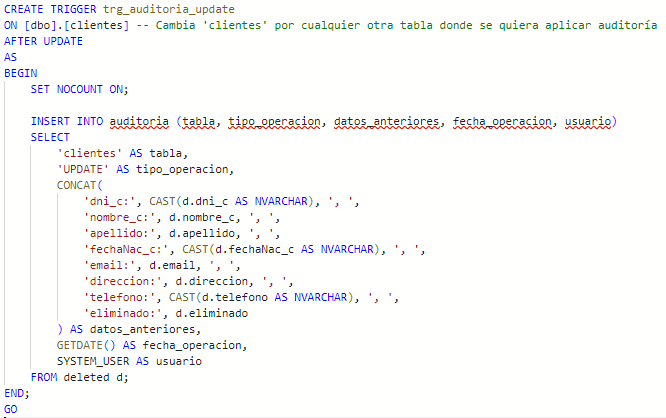
**4.4. Triggers.**

**4.4.1 Triggers de auditoría para UPDATE y DELETE**

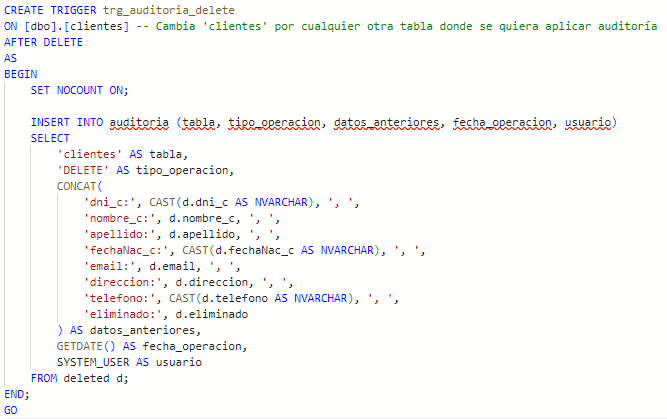
**4.4.1.1 Creamos una tabla auxiliar para registrar los cambios de auditoría**



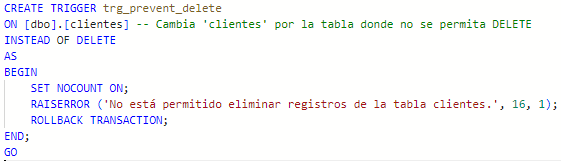
**4.4.1.2 Trigger para UPDATE. Este trigger registra los valores antes de la modificación**



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



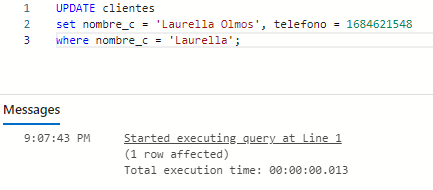
**4.4.2 Trigger para prevenir DELETE con mensaje personalizado. Este trigger previene las eliminaciones en una tabla específica y muestra un mensaje de error**



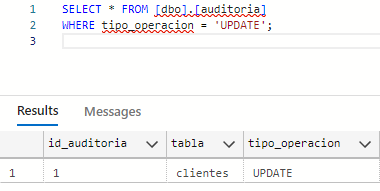
**4.4.3 Pruebas para el Trigger de Auditoría (UPDATE y DELETE)**

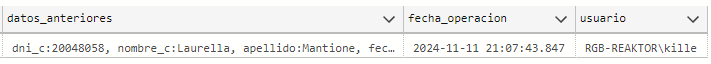
**4.4.3.1 Prueba del Trigger de Auditoría en UPDATE**

**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



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

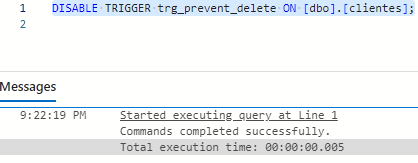




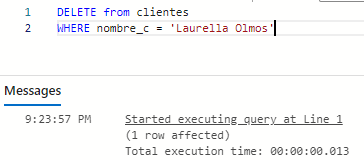
Como se puede apreciar, el update fue registrado en la tabla de auditoría con todos los datos

**4.4.3.2 Prueba del Trigger de Auditoría en DELETE**

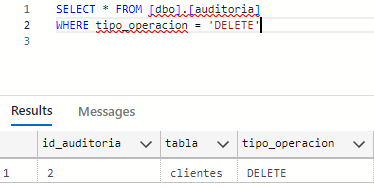
**4.4.3.2.1** Primero se desactiva el trigger de prevención de eliminación para realizar las pruebas



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



**4.4.3.2.3 Consulta la tabla auditoria para validar el registro**

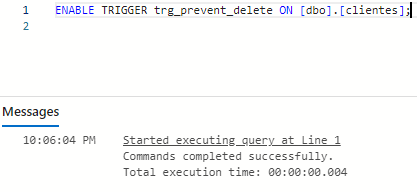




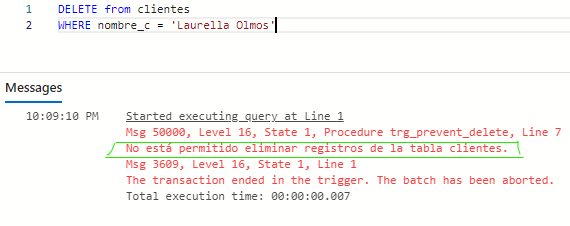
Efectivamente los cambios fueron registrados con precisión en la tabla de auditoría

**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



**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



# 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 vistos 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](https://estradawebgroup.com/Post/Cuales-son-los-diferentes-tipos-de-indices-en-SQL-Server-y-cuando-usar-cada-uno-de-ellos/20646)
* [Unidad 12. Triggers, procedimientos y funciones en MySQL - Apuntes de BD para DAW, DAM y ASIR -José Juan Sánchez Hernández - Curso 2023/2024](https://josejuansanchez.org/bd/unidad-12-teoria/index.html#triggers)
* 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](https://www.aluracursos.com/blog/que-es-y-como-trigger-en-sql)
* [Seguimiento y optimización de consultas utilizando índices SQL Server](https://www.sqlshack.com/es/seguimiento-y-optimizacion-de-consultas-utilizando-indices-sql-server/)