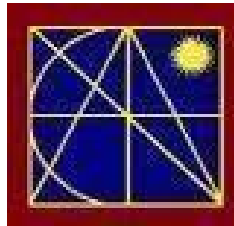




Universidad Nacional del Nordeste



Facultad de Ciencias Exactas, Naturales y Agrimensura Licenciatura en Sistemas de la
Información

BASE DE DATOS I

Diseño de Base de Datos para la Gestión de Ventas

“NikeStore”

Profesores:

Vallejos, Walter. Badaracco, Numa.

Integrantes:

- Gauna, Natalia.
- Gómez, Alina.
- Lubke, Hernán.
- Olivos Battestin, Santiago Nicolas.

Año: 2024

Índice de contenido

Contenido

1	Introducción	4
1.1	Tema	4
1.2	Definición o planteamiento del problema	4
1.3	Objetivo del proyecto	4
1.3.2	Objetivos Específicos	5
2	Marco conceptual	5
3	Metodología	
4	Desarrollo del tema / Presentación de resultados	6
4.1	Diagrama relacional	6
4.2	Diccionario de datos / tablas	6
	DICCIONARIO DE DATOS	6
5	Conclusiones	16
6	Bibliografía	16

Índice de tablas

Tabla 1-	Rol	6
Tabla 2-	Permiso	7
Tabla 3-	Proveedor	7
Tabla 4-	Cliente	8
Tabla 5 -	Usuario	9
Tabla 6 -	Categoría	10
Tabla 7 -	Producto	11

Tabla 8 - Compra	12
Tabla 9 – Detalle compra	13
Tabla 10 - Venta	14
Tabla 11 – Detalle venta	15

Índice de Figuras

Figura

1:

Diagrama

Entidad-Relación

1 Introducción

1.1 Tema

El tema que abordaremos será la gestión de una base de datos para una empresa que realiza la venta de indumentaria de la marca Nike. La aplicación que utilizará la base de datos es una aplicación de escritorio.

1.2 Definición o planteamiento del problema

La empresa necesita una base de datos eficiente que permita registrar y controlar las compras que se realizan, debe también llevar un control preciso de su inventario, para así evitar errores de stock de productos. Además, debe controlar las ventas realizadas para tener un conocimiento de los ingresos obtenidos. Por otro lado, necesita proteger la información sensible en la base de datos.

1.3 Objetivo del proyecto

El objetivo del proyecto es diseñar y desarrollar una base de datos eficiente para una aplicación de ventas.

El trabajo se enfocará en crear una estructura de datos que permita el registro, control y análisis de las compras, ventas e inventario, asegurando además la integridad y seguridad de la información sensible.

Este sistema busca optimizar los procesos de gestión de la empresa, mejorar la satisfacción del cliente y garantizar una toma de decisiones basada en datos más precisos.

La base de datos permitirá realizar:

1. **Registrar y controlar compras y ventas:** Almacenar información detallada de cada compra y venta, incluyendo, clientes, productos, y montos involucrados.
2. **Gestionar el inventario:** Controlar el stock disponible, actualizando automáticamente el inventario con cada venta realizada, evitando errores de stock.
3. **Automatizar procesos:** Simplificar y automatizar tareas relacionadas con la gestión de productos, precios, transacciones, y clientes, mejorando la eficiencia operativa.

1.3.1 Objetivos Generales

La aplicación de gestión de ventas ofrecerá ventajas tanto para los empleados como para la empresa:

- **Empleados:** Acceso conveniente a información y herramientas desde cualquier

lugar, automatización de tareas administrativas, simplificación de procesos y acceso a datos relevantes en tiempo real.

- **Empresa:** Mejora de la eficiencia operativa, centralización de la información, automatización de procesos clave, análisis de datos para la toma de decisiones y optimización de recursos.

1.3.2 Objetivos Específicos

Se espera que la aplicación de gestión empresarial:

1. **Mejore la eficiencia operativa:** Automatizando procesos rutinarios y reduciendo el tiempo dedicado a tareas administrativas.
2. **Aumente la productividad:** Proporcionando herramientas que faciliten la organización y el acceso a información relevante.
3. **Optimice el uso de recursos:** Facilitando el seguimiento de inventarios, personal y recursos operativos.

En resumen, este proyecto busca aprovechar la tecnología para ofrecer una solución efectiva que optimice los procesos internos de la empresa y mejore la gestión diaria.

2 Marco conceptual

Este marco conceptual proporciona una visión general sobre el propósito, los objetivos y los conceptos clave que guiarán el diseño y la implementación de la base de datos para una aplicación de escritorio para una tienda de indumentaria Nike. La base de datos almacenará información crítica relacionada con los productos, clientes, ventas, pagos y proveedores, asegurando una gestión eficiente y segura de la tienda. Además, se incluirán medidas de seguridad y un diseño que permita la escalabilidad y el fácil mantenimiento del sistema.

Este marco conceptual establece las bases para el desarrollo de una solución que optimiza la gestión de inventarios, ventas y relaciones con los clientes, mejorando así la eficiencia operativa y la experiencia de compra en la tienda

3 Metodología

a). Descripción del Desarrollo del Proyecto

El proyecto se desarrolló utilizando una metodología incremental, comenzando con la planificación estructurada del diseño de la base de datos y, posteriormente, implementando las funcionalidades necesarias para la gestión de inventario y control de ventas de la tienda. Las etapas principales fueron las siguientes:

1. **Análisis de Requerimientos:** Se identificaron las necesidades fundamentales del sistema, tales como el registro de productos en inventario, el control de stock, la gestión de compras y ventas, y el manejo de información confidencial para la seguridad de los datos.
2. **Diseño de la Base de Datos:** Con los requerimientos claros, se construyó un modelo entidad-relación que representa las relaciones entre las principales entidades de la base de datos, tales como productos, proveedores, ventas, compras e ingresos.
3. **Implementación del Sistema:** La base de datos fue desarrollada en un sistema de gestión de bases de datos relacional. Se crearon consultas SQL para gestionar operaciones como el registro de compras y ventas, actualización de stock y generación de reportes de ingresos.
4. **Pruebas del Sistema:** Se realizaron pruebas para verificar que el sistema cumple con las funciones requeridas y mantiene la integridad de los datos. Además, se validaron reglas de negocio como la verificación de existencias antes de registrar ventas y la protección de información sensible en la base de datos.

b). Herramientas y Procedimientos Utilizados

Para el desarrollo del proyecto se utilizaron las siguientes herramientas y procedimientos:

- **Sistema de Gestión de Bases de Datos (SGBD):** Se eligió SQL server por su estabilidad y capacidad de manejar de manera estructurada los datos de la tienda.
- **Modelado de Base de Datos:** ERDplus fue empleado para diseñar el modelo relacional basado en las especificaciones derivadas de los requerimientos del sistema, los cuales fueron detallados a medida que se definía el alcance del proyecto.

· **Control de Versiones y Colaboración:** GitHub se utilizó para el control de versiones, facilitando la colaboración entre los integrantes del equipo y permitiendo compartir, revisar y fusionar cada avance en el proyecto.

Gracias a este enfoque y a las herramientas utilizadas, fue posible desarrollar una base de datos robusta para gestionar de manera eficiente el inventario y las ventas de la tienda de indumentaria, cumpliendo con los requisitos de seguridad y control de la información de la empresa.

4 Desarrollo del tema / Presentación de resultados

4.1 Diagrama relacional

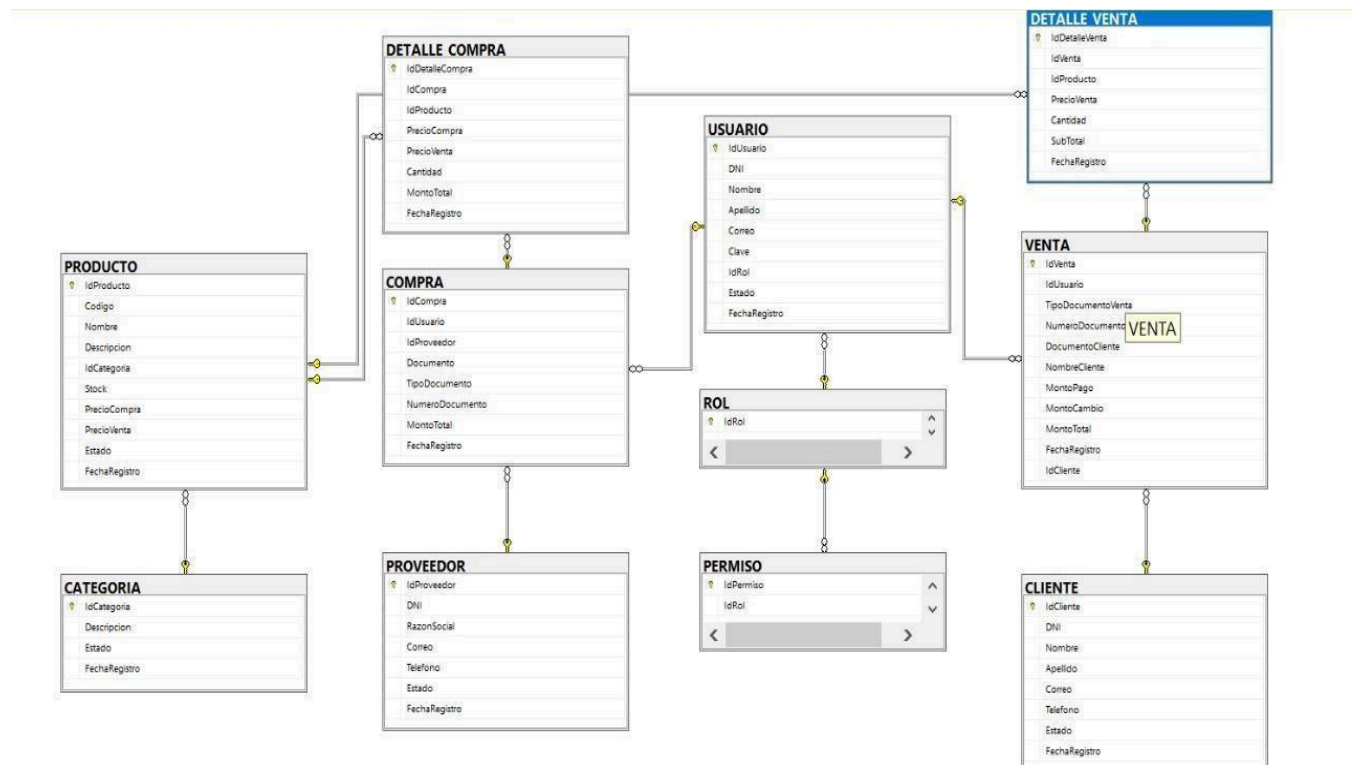


Figura 1: Diagrama Entidad-Relación

4.2 Diccionario de datos / tablas

DICCIONARIO DE DATOS

ROL					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
idRol	Identificador del rol de un usuario.	Integer		Si	Primario y key
descripción	Breve descripción del rol de Usuario	Varchar	50	No	

PERMISO					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
idPermiso	Identificador del tipo de permiso que tiene un usuario.	Integer		Si	Primario y key
idRol	Identificador del rol de un usuario.	Integer		SI	Foreign Key

nombreMenu	Nombre identificador del menu.	Varchar	50	No	
------------	--------------------------------------	---------	----	----	--

PROVEEDOR					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
idProveedor	Identificador del proveedor.	integer		Si	Primary Key
dni	Numero de dni del proveedor.	integer		No	
RazonSocial	Nombre de la empresa del proveedor	Varchar	50	No	
Correo	Correo electrónico del proveedor	Varchar	50	No	
Telefono	Número de telefono del proveedor.	Integer	50	No	
Estado	Verificación de estado activo o	Bool		no	

	inactivo.				
FechaRegistro	Fecha del registro del proveedor por primera vez.	Date		no	

CLIENTE					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
IdCliente	Identificador del cliente.	Integer		Si	Primary Key
DNI	Número de dni del cliente.	Integer		No	
nombre	Nombre del cliente.	Varchar	50	No	
apellido	Apellido del cliente.	Varchar	50	No	
correo	Correo electronico del cliente.	Varchar	50	No	
telefono	Numero de telefono del cliente.	Varchar	50	No	
estado	Verificacion de activo o inactivo del cliente.	Bool	1	No	
FechaRegistro	Fecha de registro del cliente que compra por primera vez.	Date		No	

USUARIO					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
IdUsuario	Identificador del usuario que utilizará el sistema.	Integer		Si	Primary key
DNI	Numero de dni del usuario del sistema.	Integer		No	
nombre	Nombre del usuario.	Varchar	50	No	
apellido	Apellido del usuario.	Varchar	50	No	
Correo	Correo electronico del usuario del sistema.	Varchar	50	No	
Clave	Clave alfanumerica del usuario para ingresar al sistema.	Varchar	50	No	

IdRol	Identificador del rol de un usuario.	Integer		Si	Foreign key
estado	Verificacion de activo o inactivo del usuario del sistema.	Bool	1	No	
FechaRegistro	Fecha de registro del usuario en el sistema.	Date			

CATEGORIA					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
IdCategoria	Identificador de la categoria de un producto.	Integer		Si	Primario key
Descripcion	Nombre de la categoria de un producto.	Varchar	50	No	
Estado	Verificacion de estado activo o inactivo del producto.	Bool	1	No	
FechaRegistro	Fecha de registro del producto en el sistema.	Date		No	

PRODUCTO					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
IdProducto	Identificador de un producto en el sistema.	Integer		Si	Primary key
Codigo	Codigo del producto.	Varchar	50	No	
Nombre	Nombre del producto.	Varchar	50	No	
Descripcion	Breve descripcion del producto.	Varchar	100	No	
IdCategoria	Identificador de la categoria a la que pertenece el producto.	Integer		Si	Foreign key
Stock	Cantidad del producto disponible.	Integer		No	
PrecioCompra	Monto al cual se compró el producto.	Decimal	10.2	No	
PrecioVenta	Monto al cual se venderá el producto.	Decimal	10.2	No	

Estado	Indica si el producto se encuentra disponible.	Bool	1	No	
FechaRegistro	Fecha en la cual se registró el producto en el sistema.	Date		No	

COMPRA					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
IdCompra	Identificador de la compra de los productos en el sistema.	Integer		Si	Primary key
IdUsuario	Identificador del usuario que realizó la compra.	Integer		Si	Foreign key
IdProveedor	Identificador del proveedor que realizó la compra.	Integer		Si	Foreign key
TipoDocumento	Indica que tipo de factura se realiza al efectuar la compra.	Varchar	50	No	
NumeroDocumento	Indica que tipo de factura se realiza al efectuar la compra.				
MontoTotal	Sumatoria entre los montos de los productos	float	10,2	NO	

	adquiridos.				
FechaRegistro	Fecha en la cual se compró el producto	Date			

DETALLE COMPRA						
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE	
IdDetalleCompra	Identificación del detalle de la compra.	Integer		Si	Primary key	
IdCompra	Identificador de la compra de los productos en el sistema.	Integer		Si	Foreign key	
IdProducto	Identificador de un producto en el sistema.	Integer		Si	Foreign key	
PrecioCompra	Precio al que se adquiere el producto	Float	10,2	No		
PrecioVenta	Precio al que se prevee vender el producto.	Float	10,2	No		
Cantidad	Cantidad de productos que se adquieren	Integer		No		

subtotal	Valor que se obtiene del producto entre la cantidad de productos adquiridos y su precio de compra	Float	10,2	No	
FechaRegistro	Fecha en la que se realiza la compra	Date		No	

VENTA					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
IdVenta	Identificación de la venta a un cliente	Integer		Si	Primary Key
IdUsuario	Identificación del usuario que al que se le realiza la venta	Integer		Si	Foreign Key
TipoDocumentoVenta	Tipo de facturación que se realiza	Varchar	50	No	
NumeroDocumentoVenta	Numero identificador de la facturación	Varchar	50	No	
DocumentoCliente	Número de DNI del cliente	Integer		No	
IdCliente	Nombre del cliente al que se le realiza la venta	Integer		Si	Foreign Key
MontoPago	Valor abonado por el cliente	Float	10,2	No	

MontoCambio	Diferencia entre el valor abonado por el cliente y el monto total de la compra	Float	10,2	No	
MontoTotal	Valor que se obtiene del producto entre la cantidad de productos vendidos y su precio de venta	Float	10,2	No	
FechaRegistro	Fecha en la que se realiza la venta	Date		No	

DETALLE VENTA					
NOMBRE DEL CAMPO	DESCRIPCION	TIPO DE DATO	LONGITUD	INDICE	TIPO DE INDICE
IdDetalleVenta	Valor que identifica en detalle la venta realizada.	Integer		Si	Primary key
IdVenta	Identificador de la venta realizada	Integer		Si	Foreign key
IdProducto	Identificador del producto que se está por vender.	Integer		Si	Foreign key
PrecioVenta	Valor al que se vende el producto.	Decimal	10, 2	No	
DocumentoCliente	Dni del cliente al que se le realiza la venta.	Varchar	50	No	
Cantidad	Es el número de prendas que va a comprar el cliente	Integer		No	

SubTotal	Es el valor que se obtiene del producto del precio venta con la cantidad de prendas compradas por el cliente.	Decimal	10,2	No	
FechaRegistro	Es la fecha en la cual se registra la venta realizada	Date		No	

4.3 Procedimientos Almacenados: Hallazgos.

Un procedimiento almacenado es un conjunto de instrucciones SQL predefinidas que se almacenan y ejecutan dentro de la base de datos. Estos procedimientos permiten encapsular operaciones repetitivas en un solo bloque de código que se puede ejecutar cuando sea necesario.

Características: No devuelve un valor: Un procedimiento almacenado no está diseñado para devolver un valor directamente. En su lugar, puede realizar operaciones en la base de datos (como insertar, actualizar o eliminar datos) y, si es necesario, devolver valores a través de parámetros de salida. Puede ejecutar múltiples sentencias SQL: Un procedimiento almacenado puede contener múltiples instrucciones SQL, como SELECT, INSERT, UPDATE, DELETE, y llamadas a otros procedimientos almacenados. Tiene parámetros de entrada y salida: Un procedimiento puede aceptar parámetros de entrada (para recibir valores al momento de la ejecución) y parámetros de salida (para devolver resultados o mensajes). Ejecutado explícitamente: Para ejecutar un procedimiento almacenado, se debe llamar explícitamente desde una aplicación, script SQL o desde la propia consola de la base de datos.

Funciones Almacenadas

Una función almacenada es un bloque de código SQL predefinido que, al igual que un procedimiento, se almacena en la base de datos, pero a diferencia de los procedimientos, las funciones están diseñadas para devolver un valor al momento de su ejecución.

Características: Devuelve un valor: A diferencia de un procedimiento, una función almacenada siempre devuelve un valor, ya sea un único valor o un conjunto de datos. El valor que retorna debe estar especificado por el tipo de datos que se define al crear la función. Puede ser utilizada dentro de una consulta SQL: Las funciones almacenadas pueden ser utilizadas en consultas SELECT, INSERT, UPDATE, o WHERE, ya que siempre devuelven un valor que puede ser utilizado como parte de la operación. No tiene parámetros de salida: A diferencia de los procedimientos, las funciones solo tienen parámetros de entrada. Generalmente no tiene efectos secundarios: Las funciones deben ser idempotentes (es decir, no deben alterar el estado de la base de datos, aunque algunas bases de datos permiten efectos secundarios en funciones).

Cuándo usar cada uno? Se usa procedimientos almacenados cuando necesitas realizar acciones como insertar, actualizar, borrar, o gestionar transacciones, y no necesitas que la operación devuelva un valor directo. Se usa funciones almacenadas cuando necesitas hacer cálculos o transformaciones de datos y devolver un valor que luego pueda ser utilizado directamente en una consulta.

Para poder ver el funcionamiento de los diferentes procedimientos almacenados y de las funciones almacenadas, tuvimos que insertar datos en las siguientes tablas:

En la tabla Rol:

```
insert into ROL(descripcion) values ('Administrador')
```

```
insert into ROL(descripcion) values ('Empleado')
```

```
insert into ROL(descripcion) values ('Gerente')
```

Son los roles que van a tener los usuarios que interactúan con el sistema.

En la tabla Permiso:

```
insert into PERMISO(IdRol, NombreMenu) values
(1, 'menuUsuarios'),
(1, 'menuMantenedor'),
(1, 'menuVentas'),
(1, 'menuCompras'),
(1, 'menuClientes'),
(1, 'menuProveedores'),
(1, 'menuReportes'),
(1, 'menuBackup')
```

```
insert into PERMISO(IdRol, NombreMenu) values
(2, 'menuVentas'),
(2, 'menuCompras'),
(2, 'menuClientes'),
(2, 'menuProveedores')
```

```
insert into PERMISO(IdRol, NombreMenu) values
(3, 'menuVentas'),
(3, 'menuCompras'),
(3, 'menuClientes'),
(3, 'menuReportes')
```

Son los menús a los que van a poder acceder los usuarios dependiendo de su rol.

En la Tabla Proveedores:

```
insert into PROVEEDOR(DNI, RazonSocial, Correo, Telefono, Estado) values ('40123456', 'NikeShoes', 'nikeS@gmail.com', '0800111222', 1)
insert into PROVEEDOR(DNI, RazonSocial, Correo, Telefono, Estado) values ('35789456', 'NikeJeans', 'nikeJeans@gmail.com', '0800555666', 1)
insert into PROVEEDOR(DNI, RazonSocial, Correo, Telefono, Estado) values ('45033698', 'NikeT-Shirt', 'nikeTS@gmail.com', '0810222888', 1)
```

Son los proveedores que se van a registrar en la compra de un producto.

En la tabla usuarios:

```
insert into USUARIO(DNI, Nombre, Apellido, Correo, Clave, IdRol, Estado) values ('12345678', 'Juan', 'Rojas', 'juan@gmail.com', '789', 3, 1)
insert into USUARIO(DNI, Nombre, Apellido, Correo, Clave, IdRol, Estado) values ('87654321', 'Jorge', 'Romero', 'jorge@gmail.com', '456', 2, 1)
insert into USUARIO(DNI, Nombre, Apellido, Correo, Clave, IdRol, Estado) values ('09876543', 'Agustin', 'Rodriguez', 'Agustin@gmail.com', '987', 2, 1)
insert into USUARIO(DNI, Nombre, Apellido, Correo, Clave, IdRol, Estado) values ('13579864', 'Maria', 'Losada', 'Maria@gmail.com', '654', 2, 1)
insert into USUARIO(DNI, Nombre, Apellido, Correo, Clave, IdRol, Estado) values ('40678956', 'Josefina', 'Alves', 'Josefina@gmail.com', '321', 2, 1)
insert into USUARIO(DNI, Nombre, Apellido, Correo, Clave, IdRol, Estado) values ('101010', 'Admin', 'Perez', '@gmail.com', '123', 1, 1)
```

Son los usuarios registrados en el sistema, los que se encargarán de manejarlo.

En la tabla Compras insertamos datos de distintas compras para poder probar los procedimientos almacenados y las funciones almacenadas.

Luego desarrollamos los procedimientos almacenados, para el registro, la actualización y el borrado de los usuarios.

Una vez que insertamos los datos en las distintas tablas, lo que hicimos fue desarrollar los distintos procedimientos almacenados y las funciones almacenadas.

Procedimiento almacenado para el registro de usuario:

```

create proc sp_registrarusuario(
    @DNI varchar (50),
    @Nombre varchar (50),
    @Apellido varchar (50),
    @Correo varchar (100),
    @Clave varchar (100),
    @IdRol int,
    @Estado bit,
    @IdUsuarioResultado int output,
    @Mensaje varchar(500) output
)

as
begin

    set @IdUsuarioResultado = 0
    set @Mensaje = ''

    if not exists(select 1 from USUARIO where DNI = @DNI or Correo = @Correo)
    begin
        insert into USUARIO(DNI, Nombre, Apellido, Correo, Clave, IdRol, Estado) values
        (@DNI, @Nombre, @Apellido, @Correo, @Clave, @IdRol, @Estado);

        set @IdUsuarioResultado = SCOPE_IDENTITY()
        set @Mensaje = 'Usuario registrado correctamente.';
    end
    else
    begin
        SET @Mensaje = 'El DNI o el Correo ya están registrados para otro usuario.';
    end
end

go

```

En el procedimiento almacenado para registrar un usuario, realiza la inserción de los datos solo si no existe ese DNI que se está ingresando y si no se repite el correo.

Ejecución del procedimiento almacenado para registro de usuario:

```
--Ejecucion del procedimiento registrar usuario
DECLARE @IdUsuarioResultado INT;
DECLARE @Mensaje VARCHAR(500);

exec sp_registrarusuario
@DNI = '11223344',
@Nombre = 'Nicolas',
@Apellido = 'Battestin',
@Correo = 'nicoOli@gmail.com',
@Clave = '555',
@IdRol = 2,
@Estado = 1,
@IdUsuarioResultado = @IdUsuarioResultado OUTPUT,
@Mensaje = @Mensaje output;

select @IdUsuarioResultado as ResultadoUsuario, @Mensaje as Mensaje
```

Procedimiento almacenado para el update de los datos de un usuario:

```
create proc sp_editarusuario(
    @IdUsuario int,
    @DNI varchar (50),
    @Nombre varchar (50),
    @Apellido varchar (50),
    @Correo varchar (100),
    @Clave varchar (100),
    @IdRol int,
    @Estado bit,
    @Respuesta bit output,
    @Mensaje varchar(500) output
)
as
begin

    set @Respuesta = 0
    set @Mensaje = ''

    if not exists(select 1 from USUARIO where DNI = @DNI and IdUsuario != @IdUsuario)
    begin
        update USUARIO set
            DNI = @DNI,
            Nombre = @Nombre,
            Apellido = @Apellido,
            Correo = @Correo,
            Clave = @Clave,
            IdRol = @IdRol,
            Estado = @Estado
        where IdUsuario = @IdUsuario
        set @Respuesta = 1
        set @Mensaje = 'Datos del usuario editados con exito.'
    end
    else
    begin
        set @Mensaje = 'No se puede repetir el Nro de Documento para mas de un usuario.'
    end
end
```

El procedimiento almacenado para editar los datos de un usuario pide el ingreso de los datos, y si cumple con la condición de que exista el DNI ingresado y el id usuario es diferente al ingresado, posibilita la edición de las mismas.

Ejecución del procedimiento almacenado para editar los datos de los usuario:

```
-- Ejecucion del procedimiento Editar usuario

DECLARE @Respuesta BIT;
DECLARE @Mensaje VARCHAR(500);

exec sp_editarusuario
@IdUsuario = 9,
@DNI = '101010',
@Nombre = 'Santiago',
@Apellido = 'Perez' ,
@Correo = 'Santi@gmail.com',
@Clave = 123 ,
@IdRol = 1,
@Estado = 1,
@Respuesta = @Respuesta output,
@Mensaje = @Mensaje output;

select @Respuesta as Respuesta, @Mensaje as Mensaje
```


Procedimiento almacenado para la eliminación de un usuario:

```
create proc sp_EliminarUsuario(
    @IdUsuario int,
    @Respuesta bit output,
    @Mensaje varchar(500) output
)
as
begin
    set @Respuesta = 0
    set @Mensaje = ''
    declare @pasoRegla bit = 1
    if exists (select 1
    from COMPRA c
    inner join USUARIO u on u.IdUsuario = c.IdUsuario
    where u.IdUsuario = @IdUsuario)
    begin
        set @pasoRegla = 0
        set @Respuesta = 0
        set @Mensaje = @Mensaje + 'No se puede eliminar el usuario seleccionado porque se encuentra relacionado a una compra.'
    end
    if(@pasoRegla = 1)
    begin
        delete from USUARIO
        where IdUsuario = @IdUsuario
        if(@@ROWCOUNT > 0)
        begin
            set @Respuesta = 1
            set @Mensaje = 'Usuario eliminado con exito.'
        end
    else
    begin
        SET @Respuesta = 0;
        SET @Mensaje = 'No se encontró el usuario para eliminar.';
    end
end
end
```

El procedimiento almacenado para eliminar un usuario, pide el id del usuario, luego hace un inner join con la tabla compra y luego verifica si el id del usuario esta registrado en una compra o no. En caso de estar registrada en una compra, no permite la eliminación de dicho usuario.

Ejecución del procedimiento almacenado de eliminación de un usuario:

```
DECLARE @Respuesta BIT;
DECLARE @Mensaje VARCHAR(500);

exec sp_EliminarUsuario
    @IdUsuario = 8,
    @Respuesta = @Respuesta output,
    @Mensaje = @Mensaje output

select @Respuesta as Respuesta, @Mensaje as Mensaje
```

Función almacenada para calcular la cantidad de facturas de tipo A:

```
--Funciones almacenadas

--Funcion Calcular cantidad de documentos de tipo FACTURA A
CREATE FUNCTION fn_CantidadTipoDocumentos()
RETURNS varchar(18)
AS
BEGIN
    DECLARE @TotalDocA varchar(18);

    -- Cuenta la cantidad de documentos del tipo FACTURA A
    SELECT @TotalDocA = COUNT(TipoDocumento)
    FROM COMPRA
    WHERE TipoDocumento = 'Factura A'
    group by TipoDocumento
    -- Retornar el total calculado
    RETURN @TotalDocA;
END;
GO

SELECT dbo.fn_CantidadTipoDocumentos() as 'Total Facturas A';

go
```

La función almacenada fn_cantidadTipoDocumentos cuenta la cantidad de facturas de tipo A y lo muestra.

Función almacenada para calcular el monto total de egresos monetarios de la empresa:

```
--Funcion Almacenada Monto total de egresos monetarios

CREATE FUNCTION fn_TotalCompras()
RETURNS DECIMAL(18, 2)
AS
BEGIN
    DECLARE @TotalCompras DECIMAL(18, 2);

    -- Sumar todos los valores de la columna 'Monto' en la tabla 'COMPRA'
    SELECT @TotalCompras = SUM(MontoTotal)
    FROM COMPRA;

    -- Retornar el total calculado
    RETURN @TotalCompras;
END;
GO

SELECT dbo.fn_TotalCompras() AS TotalDeCompras;

go
```

La función almacenada fn_totalCompras suma el total de montos de una compra.

Función almacenada que trae los distintos usuarios por su rol:

```
-- Funcion almacenada que trae usuarios por su rol

CREATE FUNCTION fn_ObtenerUsuariosPorRol
(
    @IdRol INT -- Parámetro de entrada para el rol
)
RETURNS TABLE
AS
RETURN
(
    SELECT IdUsuario, DNI, Nombre, Apellido, Correo, Estado
    FROM USUARIO
    WHERE IdRol = @IdRol -- Filtra por el rol especificado
);

GO

SELECT *
FROM dbo.fn_ObtenerUsuariosPorRol(2); -- Devuelve todos los usuarios con el rol de ID = 2

GO
```

La función almacenada fn_obtenerUsuariosPorRol retorna los usuarios por el idRol.

4.4 Optimización de consultas.

Índices y optimización en SQL

Los índices de SQL Server son estructuras de datos que se crean en una tabla de base de datos para mejorar la velocidad de las consultas, así como para recuperar datos más rápido y reducir los cuellos de botella que afectan a los recursos críticos. En una tabla de base de datos, los índices sirven como técnica de optimización del rendimiento. Hay diferentes tipos de índices que se pueden utilizar en SQL, pero los más comunes son:

1. Índice de Clave Primaria: Se crea automáticamente cuando se define una clave primaria en una tabla. Garantiza la unicidad de los valores en la columna de clave primaria y acelera la búsqueda por ese campo.

```
CREATE TABLE tipogasto (idtipogasto int primary key,
                        descripcion varchar(50))
go
```

2. Índice de Clave Externa: Se crea automáticamente cuando se define una clave externa (foreign key) en una tabla. Acelera las operaciones de join entre tablas relacionadas.

```
● Create table consorcio (idprovincia int,
                        idlocalidad int,
                        idconsorcio int,
                        nombre varchar(50),
                        direccion varchar(250),
                        idzona int,
                        idconserje int,
                        idadmin int,
                        Constraint PK_consorcio PRIMARY KEY (idprovincia, idlocalidad, idconsorcio),
                        Constraint FK_consorcio_pcia FOREIGN KEY (idprovincia, idlocalidad) REFERENCES localidad(idprovincia, idlocalidad),
                        Constraint FK_consorcio_zona FOREIGN KEY (idzona) REFERENCES zona(idzona),
                        Constraint FK_consorcio_conserje FOREIGN KEY (idconserje) REFERENCES conserje(idconserje),
                        Constraint FK_consorcio_admin FOREIGN KEY (idadmin) REFERENCES administrador(idadmin)
                        )
```

3. Índice No Agrupado: Se crea explícitamente en una o varias columnas seleccionadas por el desarrollador. Mejora la velocidad de búsqueda de los datos, pero no altera el orden físico de los registros en la tabla.

```
● -- Crear un @ndice no agrupado sobre la columna 'Fecha'
CREATE NONCLUSTERED INDEX IX_Fecha NonClustered ON Registros (Fecha);
```

Índices agrupados por fecha

```
● -- Crear @ndice agrupado en la columna 'Fecha'
CREATE CLUSTERED INDEX IX_fecha ON Registros (Fecha);

● -- Repetir la consulta después de crear el @ndice agrupado
DECLARE @FechaInicio DATETIME = '2015-01-01';
DECLARE @FechaFin DATETIME = '2020-01-01';

SET STATISTICS IO ON;
SET STATISTICS TIME ON;

● SELECT *
FROM Registros
WHERE Fecha BETWEEN @FechaInicio AND @FechaFin;

SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
```

Índice agrupado con columnas adicionales

```
● -- Crear un @ndice agrupado sobre 'Fecha' y columnas adicionales como 'Monto' y 'Descripcion'
CREATE CLUSTERED INDEX IX_Fecha_Incluidas ON Registros (Fecha)
INCLUDE (Monto, Descripcion);
```

Optimización de consultas

La optimización de consultas tiene como propósito el ajuste de éstas y encontrar una manera de disminuir el tiempo de respuesta de la consulta, evitar el consumo excesivo de recursos e identificar el bajo rendimiento de la consulta.

Aquí hay algunas estrategias y técnicas comunes:

1. Utiliza cláusulas como Where para filtrar los resultados y Limit para limitar la cantidad de registros recuperados. Esto permitirá que el motor de la base de datos encuentre los datos de manera más eficiente.
2. Evitar el uso de funciones en las cláusulas Where.
3. Elige cuidadosamente el tipo de Join según las relaciones entre las tablas y la información requerida en la consulta.
4. Ajusta las estadísticas de la base de dato para estimar la cantidad de filas que cumplen ciertas condiciones.

Consultas sin índices

```

-- Consulta sin @ndice
DECLARE @FechaInicio DATETIME = '2015-01-01';
DECLARE @FechaFin DATETIME = '2020-01-01';
|
SET STATISTICS IO ON; -- Habilitar estadísticas de entrada/salida
SET STATISTICS TIME ON; -- Habilitar estadísticas de tiempo

/*DECLARE @FechaInicio DATETIME = '2015-01-01';
DECLARE @FechaFin DATETIME = '2020-01-01';*/
SELECT *
FROM Registros
WHERE Fecha BETWEEN @FechaInicio AND @FechaFin;

SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;

```

Consultas con índices agrupados con columnas adicionales

```

-- Crear un @ndice agrupado sobre 'Fecha' y columnas adicionales como 'Monto' y 'Descripcion'
CREATE CLUSTERED INDEX IX_Fecha_Incluidas ON Registros (Fecha)
INCLUDE (Monto, Descripcion);

-- Repetir la consulta despu@s de crear el @ndice agrupado con columnas incluidas

SET STATISTICS IO ON;
SET STATISTICS TIME ON;
DECLARE @FechaInicio DATETIME = '2015-01-01';
DECLARE @FechaFin DATETIME = '2020-01-01';
SELECT *
FROM Registros
WHERE Fecha BETWEEN @FechaInicio AND @FechaFin;

SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;

```

Mostrar el plan de ejecución

```

-- Mostrar plan de ejecución
SET SHOWPLAN_ALL ON;

-- Realizar consulta de prueba
DECLARE @FechaInicio DATETIME = '2015-01-01';
DECLARE @FechaFin DATETIME = '2020-01-01';

SELECT *
FROM Registros
WHERE Fecha BETWEEN @FechaInicio AND @FechaFin;

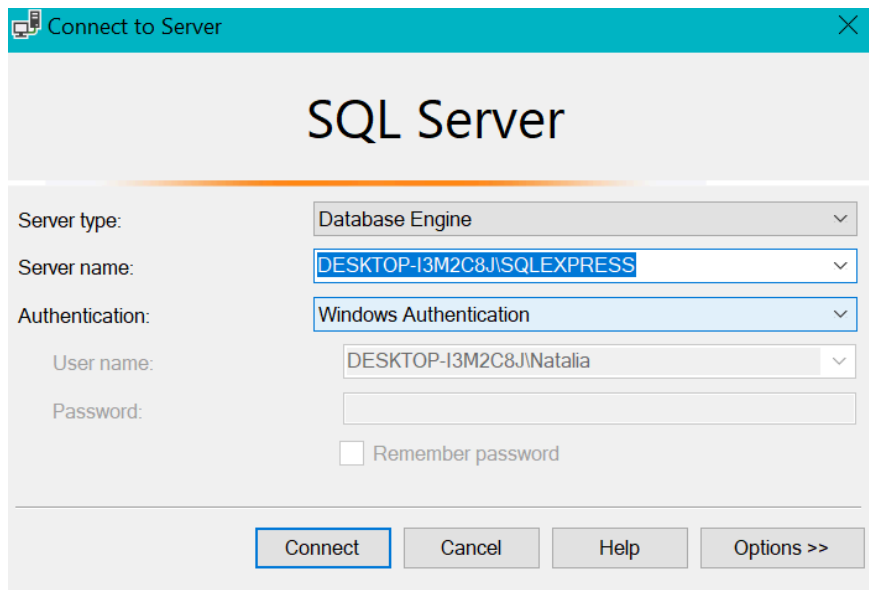
SET SHOWPLAN_ALL OFF;

```

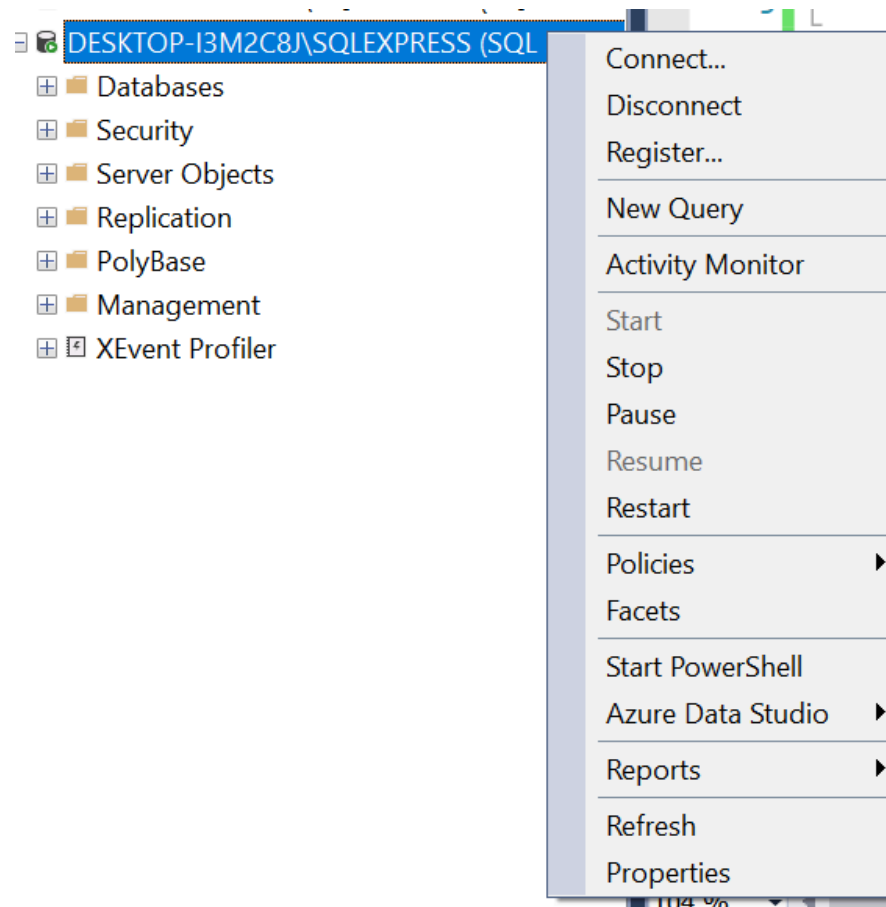
4.5 Manejo de Permisos a nivel de usuarios.

Base de datos a modo mixto:

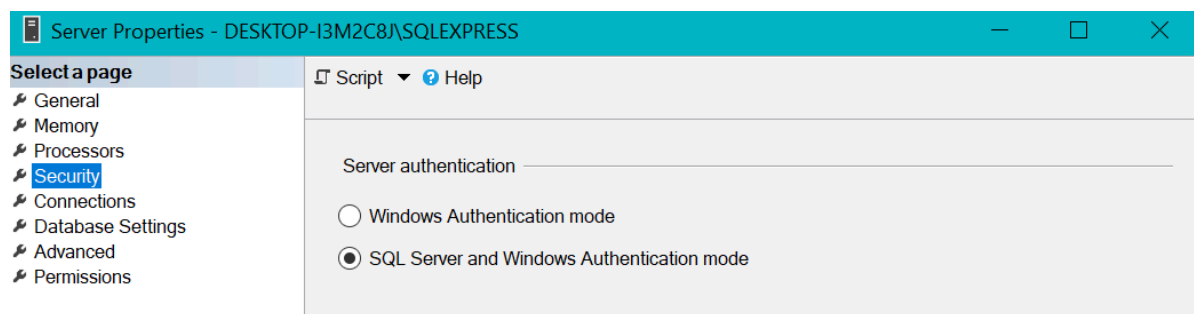
Para poder autenticar el inicio de sesión con el modo de autenticación de Windows o con la autenticación de SQL Server, se debe configurar la base de datos a modo mixto, para ello debemos iniciar sesión en SQL Server Management Studio con una cuenta que tenga permisos de administrador en el servidor.



Dirigirse a la pestaña de seguridad del servidor, dentro de la pestaña propiedades.

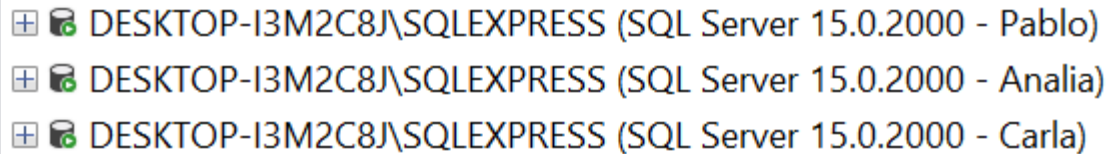


y en la sección de Autenticación de servidor, selecciona Modo de autenticación de SQL Server y de Windows.



Creacion de Login y Usuarios:

Con la sentencia "create login NOMBRE with password = ''" creamos un login y al que se le asignará un usuario con la sentencia " create user NOMBRE for login NOMBRE".

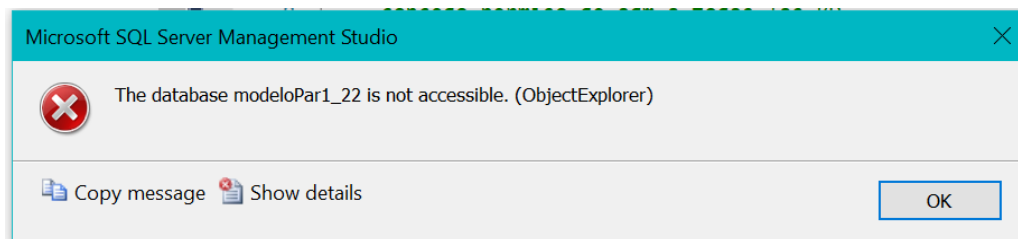


Permisos a usuarios nivel de base de datos:

Desde la cuenta de sql server con permisos de administrador, le podremos dar permisos a estos usuarios creados, ya sea a nivel de servidor, es decir para que el nuevo usuario pueda ver y editar todas las bases de datos existentes, o a nivel de base de datos, es decir, el usuario podrá acceder únicamente a la base de datos asignada. Esto podemos lograrlo a través de una serie de permisos por ejemplo SELECT, INSERT, UPDATE, DELETE en tablas específicas o en toda la base de datos.

Los permisos de los roles de base de datos definidos por el usuario se pueden personalizar con las instrucciones GRANT, DENY y REVOKE.

El usuario sin los permisos necesarios, no podrá acceder a las bases de datos.



Creación de Roles:

Al crear un rol, podremos asignar varios permisos a la vez al agregar un usuario a ese rol, en lugar de otorgar permisos uno por uno.

4.6 Manejo de datos de formato Json

¿Qué son los datos JSON?

El formato JSON (JavaScript Object Notation) es un formato que se utiliza como alternativa para la transferencia de datos estructurados entre un servidor de Web y una aplicación Web

Las partes principales que componen un JSON son las claves y los valores. Juntos forman un par clave/valor.

Podemos decir que un JSON es un objeto por la forma en la que se representa, en pares CLAVE-VALOR.

Ejemplo:

```
"Nombre": "Carlos" -> clave = Nombre, Valor= Carlos
```

Características de JSON

Al ingresar los datos JSON tenemos como resultado una sola columna que contiene todos los datos.

Si el conjunto de resultados es pequeño, puede contener una sola fila.

Si el conjunto de resultados es grande, la cadena JSON larga se divide en varias filas.

110 %

Results Messages

ID	DatosDelUsuario
1	{ "Nombre": "Cecilia", "Edad": 41, "Ciudad": "Córdoba", "Correo": "cecilia@example.com" }
2	{ "Nombre": "Luis", "Edad": 69, "Ciudad": "La Plata", "Correo": "luis@example.com" }
3	{ "Nombre": "Valeria", "Edad": 23, "Ciudad": "San Juan", "Correo": "valeria@example.com" }
4	{ "Nombre": "Raúl", "Edad": 30, "Ciudad": "Córdoba", "Correo": "raul@example.com" }
5	{ "Nombre": "Carla", "Edad": 63, "Ciudad": "Resistencia", "Correo": "carla@example.com" }
6	{ "Nombre": "Javier", "Edad": 18, "Ciudad": "Santiago del Estero", "Correo": "javier@example.com" }
7	{ "Nombre": "Paola", "Edad": 59, "Ciudad": "Mendoza", "Correo": "paola@example.com" }
8	{ "Nombre": "Juan", "Edad": 30, "Ciudad": "Mar del Plata", "Correo": "juan@example.com" }
9	{ "Nombre": "Gabriel", "Edad": 25, "Ciudad": "San Juan", "Correo": "gabriel@example.com" }
10	{ "Nombre": "Esteban", "Edad": 51, "Ciudad": "San Juan", "Correo": "esteban@example.com" }
11	{ "Nombre": "Miguel", "Edad": 51, "Ciudad": "Resistencia", "Correo": "miguel@example.com" }
12	{ "Nombre": "Clara", "Edad": 57, "Ciudad": "Corrientes", "Correo": "clara@example.com" }
13	{ "Nombre": "Cecilia", "Edad": 60, "Ciudad": "Buenos Aires", "Correo": "cecilia@example.com" }
14	{ "Nombre": "Adrian", "Edad": 38, "Ciudad": "Neuquén", "Correo": "adrian@example.com" }
15	{ "Nombre": "Adrian", "Edad": 58, "Ciudad": "San Juan", "Correo": "adrian@example.com" }

Query executed successfully.

DESKTOP-03GTKHL\HERNAN (15... | DESKTOP-03GTKHL\herna ... | ProcesamientoDatosJSON 00:00:00 | 1,500 rows

Almacenamiento

Se pueden almacenar datos JSON en columnas VARCHAR o NVARCHAR, y se indexan como texto sin formato.

Ejemplo:

```
(N'{ "Nombre": "Juan", "Edad": 30, "Ciudad": "Mar del Plata", "Correo": "juan@example.com" });
```

Procesamiento

Las funciones JSON nativas de SQL Server permiten leer, modificar, y consultar datos JSON directamente dentro de la base de datos, sin necesidad de convertir el JSON en una estructura de tabla primero.

Estas funciones son:

OPENJSON: Esta función descompone los datos en filas y columnas.

```

--Muestra un JSON en filas
SELECT
    Id,
    jsonData.[key] AS Clave,
    jsonData.[value] AS Valor
FROM Usuarios
CROSS APPLY OPENJSON(DatosDelUsuario) AS jsonData; --OPENJSON convierte el contenido de JSON en una tabla temporal

```

JSON_VALUE: Devuelve datos de un campo específico.

```

--Consultar datos JSON
SELECT
    JSON_VALUE(DatosDelUsuario, '$.Nombre') AS Nombre,
    JSON_VALUE(DatosDelUsuario, '$.Edad') AS Edad,
    JSON_VALUE(DatosDelUsuario, '$.Correo') AS Correo
FROM Usuarios;

```

JSON_QUERY: Extrae los datos completos manteniendo el formato JSON.

```

-----Consultas-----

--Consulta para datos anidados
SELECT
    JSON_QUERY(DatosDelUsuario, '$') AS DatosCompleto --JSON_QUERY es útil para obtener un objeto o arreglo JSON completo dentro de un JSON más grande
FROM Usuarios;

```

JSON_MODIFY: Modifica los datos de la columna JSON.

```
--Actualización de datos
UPDATE Usuarios
SET DatosDelUsuario = JSON_MODIFY(DatosDelUsuario, '$.Ciudad', 'Jujuy')
WHERE id = 5;
/*WHERE JSON_VALUE(DatosDelUsuario, '$.Correo') = 'carla@example.com';*/
```

Optimización de Operaciones con JSON

La optimización de datos JSON en SQL Server permite consultas eficientes y mejora el rendimiento mediante prácticas como el almacenamiento en columnas **NVARCHAR**, la indexación selectiva de valores con **JSON_VALUE**, y el uso de funciones específicas como **JSON_MODIFY** para actualizaciones puntuales. Evaluar los tiempos de ejecución y los planes de consulta asegura que las optimizaciones sean efectivas.

```
1 SET STATISTICS TIME ON;
2 SET STATISTICS IO ON;
3
4 SELECT JSON_VALUE(DatosDelUsuario, '$.Nombre') AS Nombre
5 FROM Usuarios
6 WHERE JSON_VALUE(DatosDelUsuario, '$.Edad') > 30;
7
8 SET STATISTICS TIME OFF;
9 SET STATISTICS IO OFF;
10
```

Results Messages

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

(1110 rows affected)
Table 'Usuarios'. Scan count 1, logical reads 39, physical reads 1, page server reads 0, read-ahead reads 37, page server read-ahead reads 0, 1

SQL Server Execution Times:
CPU time = 15 ms, elapsed time = 308 ms.

Completion time: 2024-11-12T22:32:29.9826544-03:00

El sistema realizó 39 lecturas lógicas, es decir, usó datos almacenados en memoria, y una lectura física, lo que significa que tuvo que acceder al disco. Esto puede hacer que el tiempo de respuesta sea un poco más lento. Además, hubo 37 lecturas en adelante, una técnica que SQL

Server utiliza para mejorar la eficiencia al anticiparse a los datos que necesitará.

En cuanto a los tiempos, el uso de la CPU fue mínimo (15 ms), mientras que el tiempo total de la consulta fue de 308 ms. Esto sugiere que hubo algo de espera por el acceso a recursos externos, como el disco. En general, fue una consulta rápida, aunque optimizar el acceso al disco podría ayudar a mejorar el rendimiento en este tipo de consultas.

5 Conclusiones

El trabajo en equipo para desarrollar la base de datos de gestión de ventas e inventario para una tienda de indumentaria NikeStore permitió abordar eficientemente las necesidades específicas de la empresa. A través de la colaboración, se logró diseñar un sistema de datos robusto que facilita el control de stock, el registro preciso de ventas y compras, y la seguridad de la información sensible. La participación de todos los integrantes contribuyó a un diseño integral y funcional, optimizando los procesos de la empresa y ayudando a mantener una gestión ordenada y confiable del negocio.

6 Bibliografía

- https://www.youtube.com/watch?v=_yiRnjx222k&list=PLMhqk5T3EuUZ-qagnxCtBRpY-LZybK9Sf
- <https://www.youtube.com/watch?v=pYBcC5DwJVQ>
- https://www.youtube.com/watch?v=giqIIGRG0RY&list=PL6kQim6ljTJsL9mC1dE8GDoZmFxtLU_bh
- <https://www.youtube.com/watch?v=cBwksNxEvNw>
- <https://learn.microsoft.com/es-es/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver16>
- <https://learn.microsoft.com/es-es/sql/relational-databases/security/permissions-database-engine?view=sql-server-ver16>
- <https://www.youtube.com/watch?v=XuFIATd6jvc>
- <https://learn.microsoft.com/es-es/sql/relational-databases/security/authentication-access/getting-started-with-database-engine-permissions?view=sql-server-ver16>

