



**Universidad Nacional del Nordeste**



**Facultad de Ciencias Exactas Naturales y Agrimensura**

**Carrera: Licenciatura en Sistemas de Información**

**Cátedra: Bases de Datos I**

**Proyecto StoredOps**

**“Manejo de permisos a nivel de usuarios de base de datos.”**

**“Procedimientos y funciones almacenadas”**

**“Optimización de consultas a través de índices”**

**“Manejo de transacciones y transacciones anidadas”**

**Alumnos:**

- Garcia Brenda
- Leguiza Cecilia Agustina
- Pinto Espíndola Malen Aymará
- Vargas Portillo Jonatan Ezequiel

**Profesor:** Dario Villegas

**Año:** 2024

## Tabla de contenido

Capítulo I Introducción .....	3
Introducción.....	3
Capítulo II Marco Conceptual.....	5
Capítulo III: Metodología seguida.....	6
Descripción de cómo se realizó el Trabajo Práctico .....	6
Herramientas (Instrumentos y Procedimientos) .....	6
Capítulo IV: Desarrollo de temas/ Presentación de resultados.....	8
Modelo de datos.....	8
Diccionario de datos.....	8
Script.....	25
Caso Práctico: Manejo de permisos a nivel de usuarios de base de datos. ....	31
Caso Práctico: Procedimientos y funciones almacenadas .....	36
Caso Práctico: Optimización de consultas a través de índices .....	38
Caso Práctico: Manejo de transacciones y transacciones anidadas .....	43
Capítulo V: Conclusiones.....	49
Capítulo VI. Bibliografía. ....	50

# Capítulo I Introducción

## Introducción

La gestión eficiente de la información es esencial para las concesionarias de vehículos, donde se manejan grandes volúmenes de datos relacionados con clientes, empleados, vehículos, pedidos y ventas. La dependencia de procesos manuales o sistemas no integrados genera errores, duplicación de datos y dificultades en el acceso a información clave, afectando la operación diaria. Este Trabajo Práctico propone el diseño e implementación de una base de datos relacional en SQL Server, que permita centralizar y optimizar la administración de la información, garantizando seguridad mediante roles y permisos, mejorando el rendimiento de consultas y transacciones, y asegurando la integridad de los datos. El proyecto busca responder a las necesidades inmediatas de la concesionaria, sentando las bases para una gestión más eficaz y escalable.

### Problema de Investigación

El problema que aborda este trabajo es la necesidad de una mejor gestión de la información en concesionarias de vehículos, donde se manejan grandes cantidades de datos relacionados con clientes, empleados, vehículos, pedidos y ventas. La gestión manual o a través de sistemas no integrados genera errores, duplicación de datos y dificultades en el acceso a información clave para las operaciones diarias.

El problema se puede plantear en las siguientes preguntas de investigación:

- ¿Cómo diseñar una base de datos eficiente que permita la gestión centralizada de clientes, empleados, pedidos y ventas?
- ¿Cómo aplicar roles y permisos a nivel de usuarios para garantizar la seguridad y confidencialidad de los datos?
- ¿Cómo mejorar el rendimiento de las consultas y transacciones en el sistema mediante la implementación de índices y optimización de consultas?

### Alcance del proyecto

El alcance del trabajo incluye el diseño, implementación y optimización de una base de datos relacional utilizando SQL Server. Se abordarán las áreas esenciales de la concesionaria, tales como la gestión de clientes, empleados, vehículos, pedidos y ventas, y se incluirán mecanismos de seguridad y control de acceso. El sistema no abordará aspectos adicionales como la integración con sistemas externos o la implementación de interfaces de usuario avanzadas. El enfoque principal es la creación de la estructura de base de datos y la eficiencia en la gestión de la información.

### Objetivo General

Desarrollar una base de datos relacional para una concesionaria de vehículos que permita gestionar de manera eficiente las operaciones de la empresa, incluyendo la administración de empleados, clientes, vehículos, pedidos y ventas. El sistema será implementado utilizando SQL Server, aplicando conceptos de roles, permisos, procedimientos almacenados y optimización de consultas.

### Objetivos Específicos

1. Diseñar un modelo de datos que integre todas las entidades relevantes de la concesionaria, tales como clientes, empleados, vehículos, pedidos y facturas.

2. Implementar procedimientos almacenados y funciones para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre los datos de la concesionaria.
3. Aplicar roles y permisos de usuarios para garantizar la seguridad de los datos, permitiendo el acceso controlado a información confidencial según el perfil del usuario.
4. Optimizar las consultas a través de la creación de índices adecuados que mejoren el rendimiento del sistema y reduzcan los tiempos de respuesta en operaciones complejas.
5. Validar la integridad de los datos mediante la implementación de restricciones y relaciones entre tablas, garantizando la consistencia del modelo.

## Capítulo II Marco Conceptual

En el ámbito de las bases de datos, la gestión eficiente y segura de la información requiere un entendimiento claro de varios conceptos fundamentales que guían su diseño y operación. A continuación, se presenta un marco conceptual que abarca los temas de manejo de permisos a nivel de usuarios, procedimientos y funciones almacenadas, optimización de consultas mediante índices y manejo de transacciones, incluyendo las transacciones anidadas.

### **Manejo de permisos a nivel de usuarios**

La administración de permisos en una base de datos implica definir y gestionar los derechos de acceso y las restricciones asignadas a diferentes usuarios o roles. Esto se realiza mediante mecanismos como privilegios (SELECT, INSERT, UPDATE, DELETE) y roles predefinidos o personalizados, que garantizan la protección de los datos frente a accesos no autorizados. Este concepto es fundamental para asegurar la integridad, confidencialidad y disponibilidad de los datos, permitiendo un control granular sobre quién puede realizar acciones específicas en la base de datos.

### **Procedimientos y funciones almacenadas**

Los procedimientos almacenados y las funciones son bloques de código predefinidos que se ejecutan en el servidor de la base de datos, optimizando el desempeño y la seguridad del sistema. Los procedimientos almacenados suelen utilizarse para ejecutar tareas complejas o recurrentes, mientras que las funciones devuelven un valor específico en función de parámetros de entrada. Ambos elementos promueven la reutilización del código, reducen la carga de la red y garantizan que las reglas de negocio se ejecuten de manera centralizada y consistente.

### **Optimización de consultas a través de índices**

Los índices son estructuras auxiliares que mejoran la velocidad de acceso a los datos en una base de datos, reduciendo el tiempo de búsqueda para consultas frecuentes. Al crear índices en columnas clave, se optimizan operaciones como búsquedas, filtros y ordenamientos. Sin embargo, un uso excesivo o inapropiado de índices puede afectar el rendimiento en operaciones de escritura, como inserciones o actualizaciones. El diseño adecuado de índices balancea eficiencia y costo computacional.

### **Manejo de transacciones y transacciones anidadas**

Las transacciones son unidades de trabajo indivisibles que aseguran la consistencia de la base de datos mediante las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Dentro de este marco, las transacciones anidadas permiten manejar operaciones dependientes entre sí, encapsulando transacciones menores dentro de una principal. Este enfoque es útil en procesos complejos, ya que facilita el manejo de errores y garantiza que los cambios sean reversibles si una parte del proceso falla, sin comprometer la integridad general de los datos.

En conjunto, estos conceptos forman la base para un manejo eficiente y seguro de bases de datos modernas, impactando directamente en la calidad del desarrollo de sistemas de información y su alineación con objetivos organizacionales.

# Capítulo III: Metodología seguida

## Descripción de cómo se realizó el Trabajo Práctico

Para la realización de este trabajo, se siguieron diversos pasos que permitieron implementar y evaluar conceptos clave de bases de datos en SQL Server, tales como el manejo de permisos a nivel de usuario, el uso de procedimientos y funciones almacenadas, la optimización de consultas a través de índices y el manejo de transacciones.

### Pasos Seguidos:

1. **Configuración de Usuarios y Permisos:** El primer paso consistió en la creación y configuración de usuarios en la base de datos, asignándoles permisos específicos según el rol que desempeñan (Administrador, Vendedor y Cajero). Esto se hizo a través de comandos SQL como CREATE LOGIN, CREATE USER, y GRANT, siguiendo una estructura de roles que permite administrar el acceso a diferentes funcionalidades de la base de datos de manera eficiente.
2. **Implementación de Procedimientos y Funciones Almacenadas:** Posteriormente, se implementaron procedimientos y funciones almacenadas que permiten realizar operaciones específicas en la base de datos de forma modular y reutilizable. Se diseñaron procedimientos que permiten consultar, insertar y manipular datos, y funciones que procesan y devuelven valores específicos dentro de consultas SQL.
3. **Optimización de Consultas mediante Índices:** Para mejorar la eficiencia de las consultas, se crearon índices en columnas clave de las tablas. Esto incluyó tanto índices agrupados como índices de cobertura en columnas utilizadas frecuentemente en las consultas. El impacto de estos índices fue evaluado mediante pruebas de rendimiento antes y después de su implementación.
4. **Implementación de Transacciones y Manejo de Errores:** Por último, se implementaron transacciones y transacciones anidadas para asegurar la atomicidad y consistencia de las operaciones en la base de datos. Se usaron comandos como BEGIN TRANSACTION, COMMIT, y ROLLBACK para controlar la ejecución de conjuntos de operaciones y garantizar la reversibilidad en caso de errores.

**Dificultades Encontradas:** Durante la implementación, se presentaron algunas dificultades en la configuración de permisos detallados para usuarios específicos, así como en el ajuste de índices que no afectaran negativamente el rendimiento de las inserciones. La integración de transacciones anidadas también requirió ajustes debido a los límites en la implementación de SQL Server para transacciones anidadas.

## Herramientas (Instrumentos y Procedimientos)

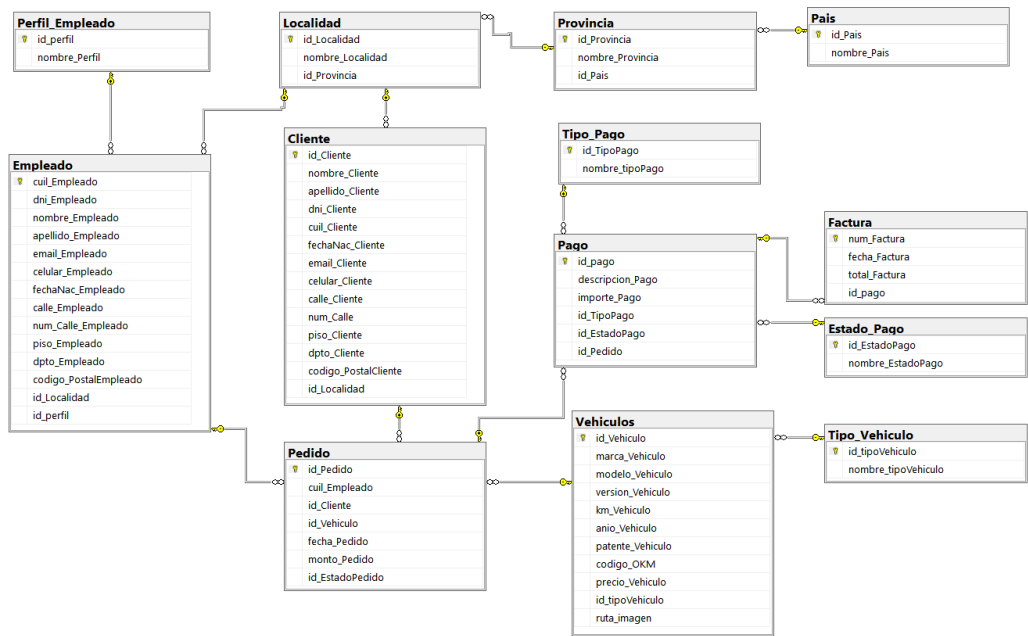
### Instrumentos y Procedimientos Utilizados:

1. **SQL Server Management Studio (SSMS):** Herramienta principal utilizada para gestionar y probar los comandos SQL implementados, incluyendo la creación de usuarios, configuración de permisos, y optimización de índices.

2. **Consultas SQL y Scripts:** Se desarrollaron scripts para automatizar la creación de usuarios, asignación de roles, y configuración de permisos. Además, se escribieron procedimientos almacenados y funciones para facilitar la gestión de los datos de la base de datos de la concesionaria.
3. **Pruebas de Rendimiento:** Se realizaron pruebas antes y después de la creación de índices para medir los tiempos de respuesta en consultas, lo cual permitió evaluar la efectividad de los índices.
4. **Método de Revisión Bibliográfica:** La consulta de documentación oficial y guías sobre SQL Server fue fundamental para comprender las mejores prácticas y aplicarlas correctamente en cada etapa del proyecto.

# Capítulo IV: Desarrollo de temas/ Presentación de resultados

## Modelo de datos



## Diccionario de datos

Características de la tabla			
Nombre		País	
Descripción		La tabla almacena información sobre los países	
Características de los datos			
campo	tipo	long	significado
id_pais	int	4	número de identificación del país
nombre_Pais	varchar	50	nombre del país
Restricciones			



Campo	Tipo de restricción
id_pais	clave primaria

Características de la tabla			
Nombre		Provincia	
Descripción		La tabla almacena información sobre las provincias	
Características de los datos			
campo	tipo	long	significado
id_Provincia	int	4	número de identificación de la provincia
nombre_Provincia	varchar	50	nombre de la provincia
id_Pais	int	4	numero del país al que pertenece
Restricciones			
Campo		Tipo de restricción	
id_Provincia		clave primaria	
Clave foránea			
Campo		Entidad asociada	
id_Pais		Pais	

Características de la tabla			
Nombre		Localidad	
Descripción		La tabla almacena información sobre las localidades	
Características de los datos			
Campo	Tipo	Long	Significado
id_Localidad	int	4	numero de identificacion de la localidad
nombre_Localidad	varchar	50	nombre de la localidad
id_Ciudad	int	4	número de la ciudad a la que pertenece
Restricciones			
Campo		Tipo de restricción	
id_Localidad		clave primaria	
Clave foránea			
Campo		Entidad asociada	
id_Ciudad		Ciudad	

Características de la tabla	
Nombre	Estado Cliente
Descripción	La tabla almacena información sobre el estado del cliente

Características de los datos			
Campo	Tipo	Long	Significado
id_Estado_Cliente	int	4	número de identificación del tipo de estado
nombre_EstadoCliente	varchar	50	nombre del tipo de estado
Restricciones			
Campo		Tipo de restricción	
id_Estado_Cliente		clave primaria	

Características de la tabla			
Nombre		Cliente	
Descripción		La tabla almacena información sobre el cliente	
Características de los datos			
campo	tipo	long	significado
id_Cliente	int	4	número de identificación del cliente
cuil_CFinal	varchar	50	almacena cuil del cliente
dni_CFinal	varchar	50	almacena dni del cliente
nombre_CFinal	varchar	50	nombre del cliente
apellido_CFinal	varchar	50	apellido del cliente
email_Cliente	varchar	50	guarda el email del cliente

celular_Cliente	varchar	50	guarda el celular del cliente
calle_Cliente	varchar	50	guarda el nombre de calle donde vive el cliente
num_Calle	int	20	guarda el numero de calle del cliente
piso_Cliente	int	10	guarda el piso donde vive el cliente
dpto_Cliente	int	10	guarda el número de dpto donde vive el cliente
codigo_PostalCliente	int	10	código postal de la ciudad a la que pertenece el cliente
id_Estado_Cliente	int	4	número de identificación del estado del cliente
id_Localidad	int	4	número de identificación de la localidad

#### Restricciones

Campo	Tipo de restricción
id_Cliente	clave primaria
email_Cliente	clave única

#### Clave foránea

Campo	Entidad asociada
id_Estado_Cliente	Estado_cliente
id_Localidad	Localidad

#### Características de la tabla

Nombre	Estado empleado
--------	-----------------

Descripción		La tabla almacena información sobre el estado del empleado	
Características de los datos			
campo	tipo	long	significado
id_Estado	int	4	número de identificación del estado
nombre_EstadoEmpleado	varchar	50	nombre del tipo de estado
Restricciones			
campo		tipo de restricción	
id_Estado		clave primaria	

Características de la tabla			
Nombre		Perfil Empleado	
Descripción		La tabla almacena información sobre el perfil del empleado	
Características de los datos			
campo	tipo	long	significado
id_perfil	int	4	número de identificación del perfil
nombre_Perfil	varchar	50	nombre del perfil
Restricciones			
campo		tipo de restricción	
id_perfil		clave primaria	

Características de la tabla			
Nombre		Empleado	
Descripción		La tabla almacena información sobre el empleado	
Características de los datos			
campo	tipo	long	significado
id_Empleado	int	7	número de identificación del empleado
nombre_Empleado	varchar	50	nombre del empleado
apellido_Empleado	varchar	60	apellido del empleado
dni_Empleado	int	8	número de dni del empleado
cuil_Empleado	int	11	cuil del empleado
celular_Empleado	varchar	20	celular del empleado
email_Empleado	varchar	100	email del empleado
calle_Empleado	varchar	100	nombre de la calle del domicilio del empleado
num_Calle_Empleado	int	4	número del domicilio del empleado
piso_Empleado	int	2	número del piso del departamento empleado
dpto_Empleado	varchar	50	nombre del departamento del empleado
codigo_PostalEmpleado	int	4	código postal del empleado
id_Localidad	int	4	numero de identificacion de la localidad

id_Perfil	int	4	número de identificación del perfil
id_Estado	int	4	número de identificación del estado
Restricciones			
campo		tipo de restricción	
id_Empleado		clave primaria	
dni_Empleado		clave única	
cuil_Empleado		clave única	
email_Empleado		clave única	
Clave foránea			
id_Localidad		clave foránea	
id_Perfil		clave foránea	
id_Estado		clave foránea	

Características de la tabla			
Nombre		Detalle_Pedido	
Descripción		La tabla almacena información sobre detalle de pedidos	
Características de los datos			
campo	tipo	long	significado

id_Vehiculo	int	4	número de identificación del vehículo
id_Pedido	int	4	número de identificación del pedido
subtotal_DetallePedido	float	10	subtotal del detalle pedido
Clave foránea			
campo		tipo de restricción	
id_Vehiculo		clave foránea	
id_Pedido		clave foránea	

Características de la tabla			
Nombre		Pedido	
Descripción		La tabla almacena información sobre los pedidos	
Características de los datos			
campo	tipo	long	significado
id_Pedido	int	4	número de identificación de pedido
id_Empleado	int	4	número de identificación del empleado
id_Cliente	int	4	número de identificación del cliente
num_Factura	int	4	número de identificación de la factura
Restricciones			
campo		tipo de restricción	



id_Pedido	clave primaria
Clave foránea	
id_Empleado	clave foránea
id_Cliente	clave foránea
num_Factura	clave foránea

Características de la tabla			
Nombre		Factura	
Descripción		La tabla almacena información sobre las facturas	
Características de los datos			
campo	tipo	long	significado
num_Factura	int	7	número de identificación de factura
fecha_Factura	date		fecha de emisión de la factura
total_Factura	float	10	monto total a pagar
id_EstadoFactura	int	4	número de identificación del estado de la factura
Restricciones			
campo		tipo de restricción	
num_Factura		clave primaria	
Clave foránea			

id_EstadoFactura	clave foránea
------------------	---------------

Características de la tabla			
Nombre		Estado_Factura	
Descripción		La tabla almacena información sobre los estados de factura	
Características de los datos			
campo	tipo	long	significado
id_EstadoFactura	int	7	número de identificación del estado de factura
nombre_EstadoFactura	varchar	20	nombre del estado de la factura
Restricciones			
campo		tipo de restricción	
id_EstadoFactura		clave primaria	

Características de la tabla	
Nombre	Pago
Descripción	La tabla almacena información sobre los pagos
Características de los datos	

campo	tipo	long	significado
id_pago	int	4	número de identificación de pago
importe_Pago	float	10	importe de pago
descripcion_Pago	varchar	50	descripción del pago
id_TipoPago	int	4	número de identificación del tipo de pago
id_EstadoPago	int	4	número de identificación del estado de pago
num_Factura	int	7	número de identificación de factura
Restricciones			
campo		tipo de restricción	
id_pago		clave primaria	
Clave foránea			
id_TipoPago		clave foránea	
id_EstadoPago		clave foránea	
num_Factura		clave foránea	

Características de la tabla	
Nombre	Tipo_Pago

Descripción		La tabla almacena información sobre los tipos de pago	
Características de los datos			
campo	tipo	long	significado
id_TipoPago	int	4	número de identificación del tipo de pago
nombre_TipoPago	varchar	20	nombre del tipo de pago
Restricciones			
campo		tipo de restricción	
id_TipoPago		clave primaria	

Características de la tabla			
Nombre		Estado_Pago	
Descripción		La tabla almacena información sobre los estados del pago	
Características de los datos			
campo	tipo	long	significado
id_EstadoPago	int	4	número de identificación del estado del pago
nombre_EstadoPago	varchar	20	nombre del tipo de pago
Restricciones			

campo	tipo de restricción
id_EstadoPago	clave primaria

Características de la tabla			
Nombre		Vehiculo	
Descripción		La tabla almacena información sobre vehículos	
Características de los datos			
campo	tipo	long	significado
id_Vehiculo	int	4	número de identificación del vehículo
marca_Vehiculo	varchar	40	marca del vehiculo
modelo_Vehiculo	varchar	20	modelo del vehículo
version_Vehiculo	varchar	20	versión del vehículo
km_Vehiculo	int	6	kilómetros del vehículo
anio_Vehiculo	int	4	año del vehículo
patente_Vehiculo	varchar	7	patente del vehículo
precio_Vehiculo	float	10	precio unitario del vehículo
id_tipoVehiculo	int	4	número de identificación del tipo de vehículo
id_Estado	int	4	número de identificación del estado del vehículo
id_Condicion	int	4	número de identificación de la condición del vehículo

Restricciones	
campo	tipo de restricción
id_Vehiculo	clave primaria
patente_Vehiculo	clave única
Clave foránea	
id_tipoVehiculo	clave foránea
id_Estado	clave foránea
id_Condicion	clave foránea

Características de la tabla			
Nombre		Tipo_Vehiculo	
Descripción		La tabla almacena información sobre los tipos de vehículos	
Características de los datos			
campo	tipo	long	significado
id_tipoVehiculo	int	4	número de identificación del tipo de vehículo
nombre_tipoVehiculo	varchar	40	nombre del tipo de vehículo
Restricciones			
campo		tipo de restricción	
id_tipoVehiculo		clave primaria	

Características de la tabla			
Nombre		Estado_Vehiculo	
Descripción		La tabla almacena información sobre los estados del vehiculo	
Características de los datos			
campo	tipo	long	significado
id_Estado	int	4	número de identificación del estado del vehículo
nombre_Estado	varchar	40	nombre del estado del vehículo
Restricciones			
campo		tipo de restricción	
id_Estado		clave primaria	

Características de la tabla			
Nombre		Condicion	
Descripción		La tabla almacena información sobre la condición del vehículo	
Características de los datos			
campo	tipo	long	significado

id_Condicion	int	4	número de identificación de la condición del vehículo
nombre_Condicion	varchar	40	nombre de la condición del vehículo
Restricciones			
campo		tipo de restricción	
id_Condicion		clave primaria	



## Script

-- DEFINNICIÓN DEL MODELO DE DATOS

CREATE DATABASE StoredOps;

GO

USE StoredOps;

GO

CREATE TABLE Pais

(  
id\_Pais INT IDENTITY(1,1) NOT NULL,  
nombre\_Pais VARCHAR(50) NOT NULL,  
CONSTRAINT PK\_Pais PRIMARY KEY (id\_Pais)  
);

CREATE TABLE Tipo\_Vehiculo

(  
id\_tipoVehiculo INT IDENTITY(1,1) NOT NULL,  
nombre\_tipoVehiculo VARCHAR(15) NOT NULL,  
CONSTRAINT PK\_TipoVehiculo PRIMARY KEY (id\_tipoVehiculo)  
);

CREATE TABLE Tipo\_Pago

(  
id\_TipoPago INT IDENTITY(1,1) NOT NULL,  
nombre\_tipoPago VARCHAR(50) NOT NULL,  
CONSTRAINT PK\_TipoPago PRIMARY KEY (id\_TipoPago)  
);

CREATE TABLE Perfil\_Empleado

```
(  
    id_perfil INT IDENTITY(1,1) NOT NULL,  
    nombre_Perfil VARCHAR(20) NOT NULL,  
    CONSTRAINT PK_Perfil PRIMARY KEY (id_perfil)  
);
```

```
CREATE TABLE Estado_Pago  
(  
    id_EstadoPago INT IDENTITY(1,1) NOT NULL,  
    nombre_EstadoPago VARCHAR(20) NOT NULL,  
    CONSTRAINT PK_EstadoPago PRIMARY KEY (id_EstadoPago)  
);
```

```
CREATE TABLE Provincia  
(  
    id_Provincia INT IDENTITY(1,1) NOT NULL,  
    nombre_Provincia VARCHAR(50) NOT NULL,  
    id_Pais INT NOT NULL,  
    CONSTRAINT PK_Provincia PRIMARY KEY (id_Provincia),  
    CONSTRAINT FK_Provincia_Pais FOREIGN KEY (id_Pais) REFERENCES  
    Pais(id_Pais)  
);
```

```
CREATE TABLE Vehiculos  
(  
    id_Vehiculo INT IDENTITY(100,1) NOT NULL,  
    marca_Vehiculo VARCHAR(20) NOT NULL,  
    modelo_Vehiculo VARCHAR(20) NOT NULL,  
    version_Vehiculo VARCHAR(20) NOT NULL,  
    km_Vehiculo INT NOT NULL,  
    anio_Vehiculo DATE NOT NULL,  
    patente_Vehiculo VARCHAR(7) NULL,
```

```
codigo_OKM INT NULL,  
precio_Vehiculo FLOAT NOT NULL,  
id_tipoVehiculo INT NOT NULL,  
ruta_imagen VARCHAR(255) NULL,  
CONSTRAINT PK_Vehiculo PRIMARY KEY (id_Vehiculo),  
CONSTRAINT FK_Vehiculos_TipoVehiculo FOREIGN KEY (id_tipoVehiculo)  
REFERENCES Tipo_Vehiculo(id_tipoVehiculo)  
);
```

```
CREATE UNIQUE INDEX UQ_Vehiculos_Patente  
ON Vehiculos(patente_Vehiculo)  
WHERE patente_Vehiculo IS NOT NULL;
```

```
CREATE UNIQUE INDEX UQ_Vehiculos_CodigoOKM  
ON Vehiculos(codigo_OKM)  
WHERE codigo_OKM IS NOT NULL;
```

```
CREATE TABLE Localidad  
(  
id_Localidad INT IDENTITY(1,1) NOT NULL,  
nombre_Localidad VARCHAR(50) NOT NULL,  
id_Provincia INT NOT NULL,  
CONSTRAINT PK_Localidad PRIMARY KEY (id_Localidad),  
CONSTRAINT FK_Localidad_Provincia FOREIGN KEY (id_Provincia) REFERENCES  
Provincia(id_Provincia)  
);
```

```
CREATE TABLE Empleado  
(  
cuil_Empleado VARCHAR(11) NOT NULL,  
dni_Empleado VARCHAR(8) NOT NULL,  
nombre_Empleado VARCHAR(50) NOT NULL,  
apellido_Empleado VARCHAR(60) NOT NULL,
```

```

email_Empleado VARCHAR(100) NOT NULL,
celular_Empleado VARCHAR(20) NOT NULL,
fechaNac_Empleado DATE NOT NULL,
calle_Empleado VARCHAR(80) NOT NULL,
num_Calle_Empleado INT NOT NULL,
piso_Empleado INT NULL,
dpto_Empleado VARCHAR(2) NULL,
codigo_PostalEmpleado INT NOT NULL,
id_Localidad INT NOT NULL,
id_perfil INT NOT NULL,
CONSTRAINT PK_Empleado PRIMARY KEY (cuil_Empleado),
CONSTRAINT FK_Empleado_Localidad FOREIGN KEY (id_Localidad)
REFERENCES Localidad(id_Localidad),
CONSTRAINT FK_Empleado_PerfilEmpleado FOREIGN KEY (id_perfil)
REFERENCES Perfil_Empleado(id_perfil),
CONSTRAINT UQ_Empleado_DniEmpleado UNIQUE (dni_Empleado),
CONSTRAINT UQ_Empleado_CuilEmpleado UNIQUE (cuil_Empleado),
CONSTRAINT UQ_Empleado_EmailEmpleado UNIQUE (email_Empleado),
CONSTRAINT CK_Empleado_DniEmpleado CHECK (dni_Empleado LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
CONSTRAINT CK_Empleado_CuilEmpleado CHECK (cuil_Empleado LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]');

```

```

CREATE TABLE Cliente

```

```

(
id_Cliente INT IDENTITY(1,1) NOT NULL,
nombre_Cliente VARCHAR(50) NOT NULL,
apellido_Cliente VARCHAR(50) NOT NULL,
dni_Cliente VARCHAR(8) NOT NULL,
cuil_Cliente VARCHAR(11) NOT NULL,
fechaNac_Cliente DATE NOT NULL,
email_Cliente VARCHAR(100) NOT NULL,
celular_Cliente VARCHAR(20) NOT NULL,

```

```

calle_Cliente VARCHAR(100) NOT NULL,
num_Calle INT NOT NULL,
piso_Cliente INT NOT NULL,
dpto_Cliente VARCHAR(2) NOT NULL,
codigo_PostalCliente INT NOT NULL,
id_Localidad INT NOT NULL,
CONSTRAINT PK_Cliente PRIMARY KEY (id_Cliente),
CONSTRAINT UQ_Cliente_Dni UNIQUE (dni_Cliente),
CONSTRAINT UQ_Cliente_Cuil UNIQUE (cuil_Cliente),
CONSTRAINT CK_Cliente_Dni CHECK (dni_Cliente LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
CONSTRAINT CK_Cliente_Cuil CHECK (cuil_Cliente LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
CONSTRAINT FK_Cliente_Localidad FOREIGN KEY (id_Localidad) REFERENCES
Localidad(id_Localidad),
CONSTRAINT UQ_Cliente_email UNIQUE (email_Cliente)
);

```

```

CREATE TABLE Pedido

```

```

(
id_Pedido INT IDENTITY(8000,1) NOT NULL,
cuil_Empleado VARCHAR(11) NOT NULL,
id_Cliente INT NOT NULL,
id_Vehiculo INT NOT NULL,
fecha_Pedido DATETIME NOT NULL,
monto_Pedido FLOAT NOT NULL,
id_EstadoPedido INT NOT NULL,
CONSTRAINT PK_Pedido PRIMARY KEY (id_Pedido),
CONSTRAINT FK_Pedido_Empleado FOREIGN KEY (cuil_Empleado)
REFERENCES Empleado(cuil_Empleado),
CONSTRAINT FK_Pedido_Cliente FOREIGN KEY (id_Cliente) REFERENCES
Cliente(id_Cliente),
CONSTRAINT FK_Pedido_Vehiculo FOREIGN KEY (id_Vehiculo) REFERENCES
Vehiculos(id_Vehiculo),

```

);

ALTER TABLE Pedido

ADD CONSTRAINT DF\_Pedido\_Fecha DEFAULT GETDATE() FOR fecha\_Pedido;

CREATE TABLE Pago

(

id\_pago INT IDENTITY(500,1) NOT NULL,

descripcion\_Pago VARCHAR(50) NULL,

importe\_Pago FLOAT NOT NULL,

id\_TipoPago INT NOT NULL,

id\_EstadoPago INT NOT NULL,

id\_Pedido INT NOT NULL,

CONSTRAINT PK\_Pago PRIMARY KEY (id\_pago),

CONSTRAINT FK\_Pago\_TipoPago FOREIGN KEY (id\_TipoPago) REFERENCES  
Tipo\_Pago(id\_TipoPago),

CONSTRAINT FK\_Pago\_EstadoPago FOREIGN KEY (id\_EstadoPago)  
REFERENCES Estado\_Pago(id\_EstadoPago),

CONSTRAINT FK\_Pago\_Pedido FOREIGN KEY (id\_Pedido) REFERENCES  
Pedido(id\_Pedido)

);

CREATE TABLE Factura

(

num\_Factura INT IDENTITY(1,1) NOT NULL,

fecha\_Factura DATE NOT NULL,

total\_Factura FLOAT NOT NULL,

id\_pago INT NOT NULL,

CONSTRAINT PK\_Factura PRIMARY KEY (num\_Factura),

CONSTRAINT FK\_Factura\_Pago FOREIGN KEY (id\_pago) REFERENCES  
Pago(id\_pago)

);

# Caso Práctico: Manejo de permisos a nivel de usuarios de base de datos.

## Permisos a Nivel de Usuarios

### Escenario

En nuestro proyecto que es acerca de una base de datos para una concesionaria, necesitamos configurar permisos para los siguientes empleados:

**Administrador:** Tiene control total sobre la base de datos (lectura, escritura, creación de tablas, etc.).

**Vendedor:** Puede consultar información de vehículos, registrar clientes y realizar pedidos.

**Cajero:** Puede gestionar los pagos y generar facturas, pero no puede modificar datos de vehículos ni clientes.

### Pasos

1. Crear los usuarios en la base de datos

```
CREATE LOGIN Admin WITH PASSWORD = 'Admin123!';
```

```
CREATE USER Admin FOR LOGIN Admin;
```

```
-- Crear usuario Vendedor
```

```
CREATE LOGIN Vendedor WITH PASSWORD = 'Vendedor123!';
```

```
CREATE USER Vendedor FOR LOGIN Vendedor;
```

```
-- Crear usuario Cajero
```

```
CREATE LOGIN Cajero WITH PASSWORD = 'Cajero123!';
```

```
CREATE USER Cajero FOR LOGIN Cajero;
```

2. Asignar permisos específicos a cada usuario

```
ALTER ROLE db_owner ADD MEMBER Admin;
```

```
-- Asignar permisos al Vendedor
```

```
GRANT SELECT ON dbo.Vehiculos TO Vendedor; -- Consultar vehículos
```

```
GRANT INSERT ON dbo.Cliente TO Vendedor; -- Registrar clientes
```

```
GRANT INSERT ON dbo.Pedido TO Vendedor; -- Crear pedidos
```

```
-- Asignar permisos al Cajero
```

```
GRANT SELECT ON dbo.Pedido TO Cajero; -- Consultar pedidos
GRANT INSERT ON dbo.Pago TO Cajero; -- Registrar pagos
GRANT INSERT ON dbo.Factura TO Cajero; -- Generar facturas
```

### 3. Pruebas de acceso

Administrador: Puede realizar cualquier operación, incluida la creación de tablas y procedimientos.

-- Crear una tabla de ejemplo (ejecutada como Admin)

```
CREATE TABLE Ejemplo_Admin (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);
```

-- Insertar un registro en la tabla Vehiculos

```
INSERT INTO Vehiculos (marca_Vehiculo, modelo_Vehiculo, version_Vehiculo,
km_Vehiculo, anio_Vehiculo, precio_Vehiculo, id_tipoVehiculo)
VALUES ('Toyota', 'Corolla', 'XLE', 15000, '2020-01-01', 20000, 1);
```

-- Eliminar un vehículo (solo el Administrador debería poder hacerlo)

```
DELETE FROM Vehiculos WHERE id_Vehiculo = 100;
```

-- Revertir el contexto al usuario original

```
REVERT;
```



```

USE StoredOps;
/* Pruebas para el Administrador (Admin)
El administrador tiene acceso completo a la base de datos, por lo que puede realizar cualquier operación:*/
-- Crear una nueva tabla (solo el Administrador debería poder hacerlo)
EXECUTE AS USER = 'Admin';

-- Crear una tabla de ejemplo (ejecutada como Admin)
CREATE TABLE Ejemplo_Admin (
    id INT PRIMARY KEY,
    nombre VARCHAR(50)
);

-- Insertar un registro en la tabla Vehiculos
INSERT INTO Vehiculos (marca_Vehiculo, modelo_Vehiculo, version_Vehiculo, km_Vehiculo, anio_Vehiculo, precio_Vehiculo, id_tipoVehiculo)
VALUES ('Toyota', 'Corolla', 'XLE', 15000, '2020-01-01', 20000, 1);

-- Eliminar un vehículo (solo el Administrador debería poder hacerlo)
DELETE FROM Vehiculos WHERE id_Vehiculo = 109;

```

100 %

Messages

(1 row affected)

(1 row affected)

Completion time: 2024-11-14T07:19:18.0949832-03:00

Vendedor: Consulta vehículos, registra clientes y genera pedidos. No puede modificar datos de pago.

USE StoredOps;

-- Consultar información de vehículos

SELECT \* FROM Vehiculos;

```

USE StoredOps;
EXECUTE AS USER = 'Vendedor';
-- Consultar información de vehículos
SELECT * FROM Vehiculos;

```

100 %

Results Messages

	id_Vehiculo	marca_Vehiculo	modelo_Vehiculo	version_Vehiculo	km_Vehiculo	anio_Vehiculo	patente_Vehiculo	codigo_OKM	precio_Vehiculo	id_tipoVehiculo	ruta_imagen
1	100	Toyota	Corolla	SE	15000	2021-01-01	ABC123	NULL	25000	1	/images/toyota_corolla.png
2	101	Ford	Ecosport	Titanium	30000	2020-06-01	DEF456	NULL	28000	2	/images/ford_ecosport.png
3	102	Chevrolet	S10	High Country	40000	2019-03-01	GHI789	NULL	32000	3	/images/chevrolet_s10.png
4	103	Volkswagen	Golf	Comfortline	20000	2022-05-01	JKL012	NULL	27000	4	/images/vw_golf.png
5	104	Honda	Civic	EX	10000	2021-09-01	MNO345	NULL	24000	1	/images/honda_civic.png

-- Registrar un nuevo cliente

INSERT INTO Cliente (nombre\_Cliente, apellido\_Cliente, dni\_Cliente, cuil\_Cliente, fechaNac\_Cliente, email\_Cliente, celular\_Cliente, calle\_Cliente, num\_Calle, piso\_Cliente, dpto\_Cliente, codigo\_PostalCliente, id\_Localidad)

VALUES ('Carlos', 'Gomez', '37951852', '20379518521', '1985-05-05', 'carlos.gomez1@cliente.com', '3795185252', 'Calle Falsa', 123, 1, 'A', 5001, 2);

-- Crear un nuevo pedido

```
INSERT INTO Pedido (cuil_Empleado, id_Cliente, id_Vehiculo, fecha_Pedido,
monto_Pedido, id_EstadoPedido)
```

```
VALUES
```

```
('20304050607', 1, 100, '2023-01-01', 25500, 1);
```

```
-- Intentar eliminar un vehículo (esto debería ser denegado)
```

```
DELETE FROM Vehiculos WHERE id_Vehiculo = 101; -- Espera un error de permisos
```

```
/* Pruebas para el Vendedor
El vendedor tiene permisos para consultar información de vehículos, registrar clientes y crear pedidos,
pero no debería poder modificar datos de pagos ni eliminar vehículos.*/

EXECUTE AS USER = 'Vendedor';

-- Consultar información de vehículos
SELECT * FROM Vehiculos;

-- Registrar un nuevo cliente
INSERT INTO Cliente (nombre_Cliente, apellido_Cliente, dni_Cliente, cuil_Cliente, fechaNac_Cliente, email_Cliente, cel)
VALUES ('Carlos', 'Gomez', '37931852', '20379318521', '1985-05-05', 'carlos.gomez2@cliente.com', '3795185252', 'Calle')

-- Crear un nuevo pedido
INSERT INTO Pedido (cuil_Empleado, id_Cliente, id_Vehiculo, fecha_Pedido, monto_Pedido, id_EstadoPedido)
VALUES ('20304050607', 1, 100, '2023-01-01', 25500, 1);

-- Intentar eliminar un vehículo (esto debería ser denegado)
DELETE FROM Vehiculos WHERE id_Vehiculo = 101; -- Espera un error de permisos
```

100 %

Results Messages

(5 rows affected)

(1 row affected)

(1 row affected)

Msg 229, Level 14, State 6, Line 47  
Se denegó el permiso DELETE en el objeto 'Vehiculos', base de datos 'StoredOps', esquema 'dbo'.

Completion time: 2024-11-14T07:12:23.9769644-03:00

**Cajero: Genera facturas y gestiona pagos. No puede modificar vehículos ni registrar clientes.**

```
USE StoredOps;
```

```
-- Realizar un pago
```

```
INSERT INTO Pago (descripcion_Pago, importe_Pago, id_TipoPago, id_EstadoPago,
num_Factura)
```

```
VALUES ('Pago Inicial', 5000, 1, 1, 1);
```

```
-- Generar una factura
```

```
INSERT INTO Factura (fecha_Factura, total_Factura, id_EstadoFactura)
```

```
VALUES (GETDATE(), 5000, 1);
```

-- Intentar registrar un cliente (esto debería ser denegado)

```
INSERT INTO Cliente (email_Cliente, celular_Cliente, calle_Cliente, num_Calle,  
piso_Cliente, dpto_Cliente, codigo_PostalCliente, id_Estado_Cliente, id_Localidad)
```

```
VALUES ('otro_cliente@example.com', '0987654321', 'Calle Nueva', 456, 2, 'B', 5678,  
1, 1); -- Espera un error de permisos
```

```
/* Pruebas para el Cajero
El cajero tiene permisos para gestionar pagos y generar facturas, pero no puede modificar datos de vehículos ni registrar nuevos clientes.*/

EXECUTE AS USER = 'Cajero';

-- Realizar un pago
INSERT INTO Pago (descripcion_Pago, importe_Pago, id_TipoPago, id_EstadoPago, id_Pedido)
VALUES ('Pago en efectivo', 18000.00, 2, 2, 8005);

-- Generar una factura
INSERT INTO Factura (fecha_Factura, total_Factura, id_pago)
VALUES (GETDATE(), 11000.00, 1);

-- Intentar registrar un cliente (esto debería ser denegado)
INSERT INTO Cliente (nombre_Cliente, apellido_Cliente, dni_Cliente, cuil_Cliente, fechaNac_Cliente, email_Cliente, celular_Cliente, calle_Cliente, num_Calle, piso_Cliente, dpto_Cliente, codigo_PostalCliente, id_Estado_Cliente, id_Localidad)
VALUES ('Maria', 'Gonzalez', '36517962', '20365179621', '1985-05-05', 'mariagon@cliente.com', '379554321', 'Calle Falsa', 123, 1, 'A', 5001, 2);
```

10 %

Messages

(1 row affected)

(1 row affected)

Msg 229, Level 14, State 6, Line 66  
Se denegó el permiso INSERT en el objeto 'Cliente', base de datos 'StoredOps', esquema 'dbo'.

Completion time: 2024-11-14T07:10:10.1045811-03:00

## Resultados obtenidos

**Administrador:** Tiene control total sobre la base de datos. Puede crear, modificar y consultar cualquier dato.

**Vendedor:** Puede consultar vehículos y registrar clientes. Puede crear pedidos, pero no puede modificar tablas o eliminar datos.

**Cajero:** Puede gestionar pagos y generar facturas. No tiene acceso a modificar datos de vehículos ni clientes.

## Caso Práctico: Procedimientos y funciones almacenadas

### Crea un procedimiento para buscar un Vehículo determinado

```
CREATE PROCEDURE buscar_vehiculo
    @NombreBuscar VARCHAR(50),
    @Precio DECIMAL(8, 2)
AS --palabra clave
BEGIN
    SET NOCOUNT ON;      --ayuda a suprimir mensajes innecesarios sobre el
                           número de filas afectadas
    SELECT * FROM Vehiculos WHERE modelo_Vehiculo LIKE '%' +
    @NombreBuscar + '%' AND precio_Vehiculo <= @Precio;
END
```

--Ejecuta un procedimiento almacenado

EXEC buscar\_vehiculo 'Ecosport', 28000;



The screenshot displays a SQL script in the 'Script' tab of SQL Server Enterprise Manager. The script includes the creation of a stored procedure named 'buscar\_vehiculo' and its execution with parameters 'Ecosport' and 28000. Below the script, the 'Results' tab shows a single row of data from the 'Vehiculos' table.

	id_Vehiculo	marca_Vehiculo	modelo_Vehiculo	version_Vehiculo	km_Vehiculo	anio_Vehiculo	patente_Vehiculo	codigo_OKM	precio_Vehiculo	id_tipoVehiculo	ruta_imagen
1	101	Ford	Ecosport	Titanium	30000	2020-06-01	DEF456	NULL	28000	2	/images/ford_ecosport.png

**Función:** Se crea una funcion para realizar el promedio por tipo de vehículo. En este caso se busca un tipo de vehiculo específico

```
CREATE FUNCTION dbo.PrecioPromedioPorTipo (
    @idTipoVehiculo INT -- Parámetro de entrada: id del tipo de vehículo
)
RETURNS FLOAT -- retorno: el precio promedio
```

AS

BEGIN

DECLARE @PrecioPromedio FLOAT;

-- Calcular el precio promedio de los vehículos del tipo especificado

SELECT @PrecioPromedio = AVG(precio\_Vehiculo)

FROM Vehiculos

WHERE id\_tipoVehiculo = @idTipoVehiculo;

-- Devuelve el precio promedio

RETURN @PrecioPromedio;

END;

--PARA EJECUTAR

----- Ejemplo: obtener el precio promedio para el Tipo de vehículo con id = 2 - ECOSPORT

SELECT dbo.PrecioPromedioPorTipo(2) AS PrecioPromedio;

```
95
96 ---- funciones almacenadas -CREAR
97
98 CREATE FUNCTION dbo.PrecioPromedioPorTipo (
99     @idTipoVehiculo INT -- Parámetro de entrada: id del tipo de vehículo
100 )
101 RETURNS FLOAT -- retorno: el precio promedio
102 AS
103 BEGIN
104     DECLARE @PrecioPromedio FLOAT;
105     -- Calcular el precio promedio de los vehículos del tipo especificado
106     SELECT @PrecioPromedio = AVG(precio_Vehiculo)
107     FROM Vehiculos
108     WHERE id_tipoVehiculo = @idTipoVehiculo;
109     -- Devuelve el precio promedio
110     RETURN @PrecioPromedio;
111 END;
112
113 --PARA USAR
114 ----- Ejemplo: obtener el precio promedio para el Tipo de vehículo con id = 2 Ecosport
115 SELECT dbo.PrecioPromedioPorTipo(2) AS PrecioPromedio;
116
```

30 %

Results		Messages	
	PrecioPromedio		
1	28000		

# Caso Práctico: Optimización de consultas a través de índices

## Creación y Evaluación de Índices

### Escenario

Para este ejercicio, usaremos una tabla con un millón de registros. La tabla contendrá una columna de fecha sin índice inicial, donde realizaremos consultas de búsqueda por rango de fechas. Después, agregaremos distintos tipos de índices y mediremos el impacto en el rendimiento.

### Objetivos del Ejercicio

Evaluar el impacto de los índices en el rendimiento de consultas: Identificar mejoras en los tiempos de respuesta y planes de ejecución. Comparar tiempos de respuesta con y sin índices: Verificar las diferencias de rendimiento entre diferentes configuraciones de índices. **Pasos** Carga masiva de datos en la tabla Cliente:

1. Se generará una tabla de un millón de registros, que incluye una columna de tipo fecha para realizar las consultas.

```
ALTER TABLE Cliente NOCHECK CONSTRAINT ALL;
```

Script para la carga masiva de registros sobre la tabla Cliente en el campo fechaNac\_Cliente (sin índice)

```
DECLARE @i INT = 1;
```

```
WHILE @i <= 1000000
```

```
BEGIN
```

```
    INSERT INTO Cliente (
```

```
        nombre_Cliente,
```

```
        apellido_Cliente,
```

```
        dni_Cliente,
```

```
        cuil_Cliente,
```

```
        fechaNac_Cliente,
```

```
        email_Cliente,
```

```
        celular_Cliente,
```

```

    calle_Cliente,
    num_Calle,
    piso_Cliente,
    dpto_Cliente,
    codigo_PostalCliente,
    id_Localidad
)
VALUES (
    'Nombre' + CAST(@i AS VARCHAR(7)),
    'Apellido' + CAST(@i AS VARCHAR(7)),
    RIGHT('00000000' + CAST(@i AS VARCHAR(8)), 8),
    RIGHT('000000000000' + CAST(@i AS VARCHAR(11)), 11),
    DATEADD(DAY, -ABS(CHECKSUM(NEWID())) % 36500, GETDATE()), -- Genera
fechas de nacimiento aleatorias
    'email' + CAST(@i AS VARCHAR(7)) + '@example.com',
    '1234567890',
    'Calle Ficticia',
    FLOOR(RAND() * 1000),
    FLOOR(RAND() * 10),
    'A',
    1234,
    1
);

SET @i = @i + 1;

```

END;

```
-- Habilitar restricciones de integridad referencial y constraints
ALTER TABLE Cliente CHECK CONSTRAINT ALL;

--Realizar una búsqueda por periodo y registrar el plan de ejecución utilizado por el motor y los tiempos de respuesta.
SET STATISTICS TIME ON;
SET STATISTICS IO ON;

-- Consulta sin índice
SELECT *
FROM Cliente
WHERE fechaNac_Cliente BETWEEN '1990-01-01' AND '2000-12-31';

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

110 %

Results Messages

(110034 rows affected)  
Tabla "Cliente". Número de examen 1, lecturas lógicas 17890, lecturas físicas 0, lecturas de servidor de páginas 0, lecturas anticipadas 0, lecturas anticipadas de serv

Tiempos de ejecución de SQL Server:  
Tiempo de CPU = 31 ms, tiempo transcurrido = 1254 ms.

Completion time: 2024-11-14T02:41:06.6927405-03:00

2. Realizar consulta inicial sin índice: Realizamos una búsqueda por un rango de fechas y registramos el tiempo de respuesta y el plan de ejecución.

SET STATISTICS TIME ON;

SET STATISTICS IO ON;

-- Consulta sin índice

SELECT \*

FROM Cliente

WHERE fechaNac\_Cliente BETWEEN '1990-01-01' AND '2000-12-31';

SET STATISTICS TIME OFF;

SET STATISTICS IO OFF;

```
-- eliminar la restricción de clave primaria asociada al índice PK_Cliente
ALTER TABLE Cliente
DROP CONSTRAINT PK_Cliente;

-- Crear el nuevo índice clúster sobre fechaNac_Cliente
CREATE CLUSTERED INDEX IX_Cliente_FechaNac ON Cliente(fechaNac_Cliente);

-- Consulta con índice agrupado
SET STATISTICS TIME ON;
SET STATISTICS IO ON;

SELECT *
FROM Cliente
WHERE fechaNac_Cliente BETWEEN '1990-01-01' AND '2000-12-31';
```

110 %

Results Messages

(110034 rows affected)  
Tabla "Cliente". Número de examen 1, lecturas lógicas 2046, lecturas físicas 0, lecturas de servidor de páginas 0, lecturas anticipadas 0, lecturas anticipadas de servi

Tiempos de ejecución de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 1238 ms.

Completion time: 2024-11-14T02:56:42.9046859-03:00

3. Crear índice agrupado en la columna de fecha: Añadimos un índice agrupado en la columna de fecha y repetimos la consulta para medir las diferencias.



-- eliminar la restricción de clave primaria asociada al índice PK\_Cliente

ALTER TABLE Cliente

DROP CONSTRAINT PK\_Cliente;

-- Crear el nuevo índice clúster sobre fechaNac\_Cliente

CREATE CLUSTERED INDEX IX\_Cliente\_FechaNac ON Cliente(fechaNac\_Cliente);

-- Consulta con índice agrupado

SET STATISTICS TIME ON;

SET STATISTICS IO ON;

SELECT \*

FROM Cliente

WHERE fechaNac\_Cliente BETWEEN '1990-01-01' AND '2000-12-31';

SET STATISTICS TIME OFF;

SET STATISTICS IO OFF;

4. Borrar el índice creado

DROP INDEX IX\_Cliente\_FechaNac ON Cliente;

5. Crear indice no agrupado de cobertura y repetimos la consulta para medir las diferencias.

-- Crear un índice no clúster

CREATE NONCLUSTERED INDEX IX\_Cliente\_FechaNac\_Cubierto

ON Cliente(fechaNac\_Cliente)

INCLUDE (nombre\_Cliente, apellido\_Cliente, dni\_Cliente, cuil\_Cliente);

-- Consulta con índice no clúster cubierto

SET STATISTICS TIME ON;

SET STATISTICS IO ON;

-- Ejecutar la consulta que utilizará el índice cubierto

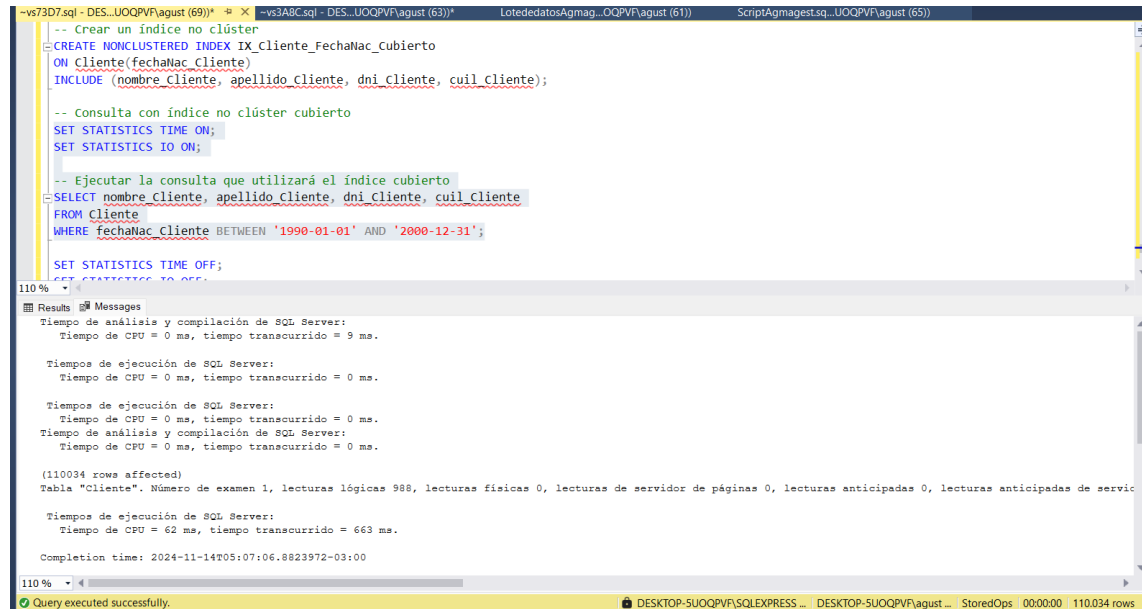
SELECT nombre\_Cliente, apellido\_Cliente, dni\_Cliente, cuil\_Cliente

FROM Cliente

```
WHERE fechaNac_Cliente BETWEEN '1990-01-01' AND '2000-12-31';
```

```
SET STATISTICS TIME OFF;
```

```
SET STATISTICS IO OFF;
```



```
-- Crear un índice no clúster
CREATE NONCLUSTERED INDEX IX_Cliente_fechaNac_Cubierto
ON Cliente(fechaNac_Cliente)
INCLUDE (nombre_Cliente, apellido_Cliente, dni_Cliente, cuil_Cliente);

-- Consulta con índice no clúster cubierto
SET STATISTICS TIME ON;
SET STATISTICS IO ON;

-- Ejecutar la consulta que utilizará el índice cubierto
SELECT nombre_Cliente, apellido_Cliente, dni_Cliente, cuil_Cliente
FROM Cliente
WHERE fechaNac_Cliente BETWEEN '1990-01-01' AND '2000-12-31';

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

Results Messages

Tiempo de análisis y compilación de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 9 ms.

Tiempo de ejecución de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.

Tiempo de ejecución de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.

Tiempo de análisis y compilación de SQL Server:  
Tiempo de CPU = 0 ms, tiempo transcurrido = 0 ms.

(110034 rows affected)

Tabla "Cliente". Número de examen 1, lecturas lógicas 988, lecturas físicas 0, lecturas de servidor de páginas 0, lecturas anticipadas 0, lecturas anticipadas de servicio 0.

Tiempo de ejecución de SQL Server:  
Tiempo de CPU = 62 ms, tiempo transcurrido = 663 ms.

Completion time: 2024-11-14T05:07:06.8823972-03:00

Query executed successfully. DESKTOP-SUOQPVF\SQLEXPRESS ... DESKTOP-SUOQPVF\agust ... StoredOps 00:00:00 110.034 rows

## Resultados Obtenidos

**Consulta sin índice:** Se evaluó que la consulta inicial sin índice tuvo un tiempo de respuesta elevado debido a la búsqueda en toda la tabla.

**Con índice agrupado:** El tiempo de respuesta mejoro significativamente, ya que los datos están ordenados físicamente por la columna de fecha.

**Índice no agrupado de cobertura** Muestra un rendimiento mejorado, ya que filtra la consulta por una fecha y sus filas fueron menores.

## Caso Práctico: Manejo de transacciones y transacciones anidadas

Escribir el código Transact SQL que permita definir una transacción consistente en: Insertar un registro en alguna tabla, luego otro registro en otra tabla y por último la actualización de uno o más registros en otra tabla. Actualizar los datos solamente si toda la operación es completada con éxito.

Sobre el código escrito anteriormente provocar intencionalmente un error luego del insert y verificar que los datos queden consistentes (No se debería realizar ningún insert).

### Ejemplo práctico aplicado:

```
BEGIN TRY
```

```
    BEGIN TRANSACTION;
```

```
        -- Insertar un nuevo país
```

```
        DECLARE @PaisID INT;
```

```
        INSERT INTO Pais (nombre_Pais) VALUES ('Chile');
```

```
        SET @PaisID = SCOPE_IDENTITY();
```

```
        -- Provocar un error después del primer insert para simular
```

```
        -- una falla (se puede comentar esta línea para probar sin errores)
```

```
        --THROW 50001, 'Error simulado después de insertar Pais', 1;
```

```
        -- Insertar una provincia relacionada con el país
```

```
        INSERT INTO Provincia (nombre_Provincia, id_Pais) VALUES  
        ('Santiago', @PaisID);
```

```
        -- Confirmar la transacción
```

```
        COMMIT TRANSACTION;
```

```
        PRINT 'Transacción completada con éxito';
```

```
    END TRY
```

```

BEGIN CATCH

    -- Revertir la transacción en caso de error
    ROLLBACK TRANSACTION;

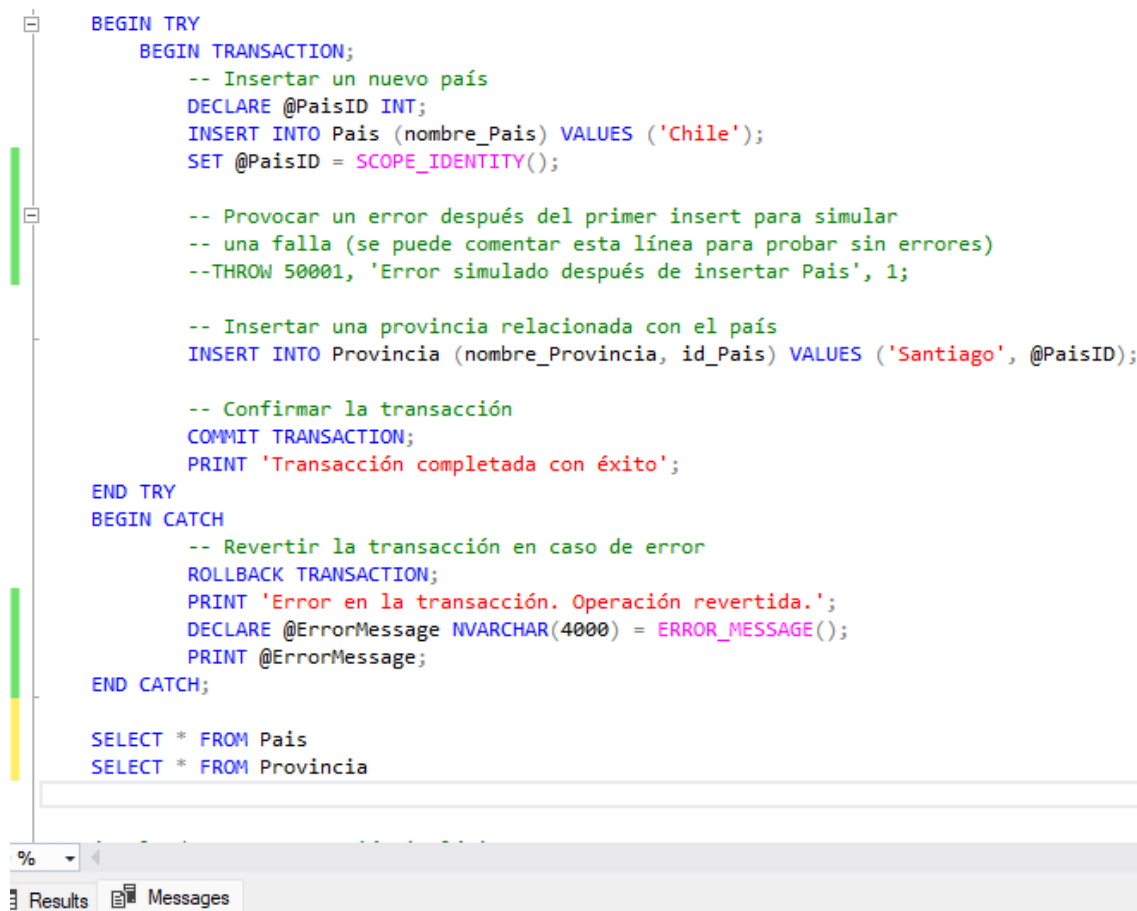
    PRINT 'Error en la transacción. Operación revertida.';

    DECLARE      @ErrorMessage      NVARCHAR(4000)      =
ERROR_MESSAGE();

    PRINT @ErrorMessage;

END CATCH;

```



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a T-SQL script with a TRY-CATCH block. The script attempts to insert a new country ('Chile') into the 'Pais' table, then a province ('Santiago') into the 'Provincia' table, and finally commits the transaction. The bottom pane shows the execution results, indicating that both insert operations were successful and the transaction was completed successfully.

```

BEGIN TRY
    BEGIN TRANSACTION;
    -- Insertar un nuevo país
    DECLARE @PaisID INT;
    INSERT INTO Pais (nombre_Pais) VALUES ('Chile');
    SET @PaisID = SCOPE_IDENTITY();

    -- Provocar un error después del primer insert para simular
    -- una falla (se puede comentar esta línea para probar sin errores)
    --THROW 50001, 'Error simulado después de insertar Pais', 1;

    -- Insertar una provincia relacionada con el país
    INSERT INTO Provincia (nombre_Provincia, id_Pais) VALUES ('Santiago', @PaisID);

    -- Confirmar la transacción
    COMMIT TRANSACTION;
    PRINT 'Transacción completada con éxito';
END TRY
BEGIN CATCH
    -- Revertir la transacción en caso de error
    ROLLBACK TRANSACTION;
    PRINT 'Error en la transacción. Operación revertida.';
    DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
    PRINT @ErrorMessage;
END CATCH;

SELECT * FROM Pais
SELECT * FROM Provincia

```

Results Messages

(1 row affected)

(1 row affected)

Transacción completada con éxito

### Resultado obtenido:

- Se logró insertar esos nuevos registros en las respectivas tablas sin ningún problema.

**Ejemplo práctico aplicado, pero con una simulación de error:**

```

BEGIN TRY
    BEGIN TRANSACTION;

        -- Insertar un nuevo país
        DECLARE @PaisID INT;
        INSERT INTO Pais (nombre_Pais) VALUES ('Chile');
        SET @PaisID = SCOPE_IDENTITY();

        -- Provocar un error después del primer insert para simular
        -- una falla (se puede comentar esta línea para probar sin errores)
        THROW 50001, 'Error simulado después de insertar Pais', 1;

        -- Insertar una provincia relacionada con el país
        INSERT INTO Provincia (nombre_Provincia, id_Pais) VALUES
('Santiago', @PaisID);

        -- Confirmar la transacción
        COMMIT TRANSACTION;
        PRINT 'Transacción completada con éxito';
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        ROLLBACK TRANSACTION;
        PRINT 'Error en la transacción. Operación revertida.';
        DECLARE @ErrorMessage NVARCHAR(4000) =
ERROR_MESSAGE();
        PRINT @ErrorMessage;
    END CATCH;

```

```
BEGIN TRY
    BEGIN TRANSACTION;
    -- Insertar un nuevo país
    DECLARE @PaisID INT;
    INSERT INTO Pais (nombre_Pais) VALUES ('Chile');
    SET @PaisID = SCOPE_IDENTITY();

    -- Provocar un error después del primer insert para simular
    -- una falla (se puede comentar esta línea para probar sin errores)
    THROW 50001, 'Error simulado después de insertar Pais', 1;

    -- Insertar una provincia relacionada con el país
    INSERT INTO Provincia (nombre_Provincia, id_Pais) VALUES ('Santiago', @PaisID);

    -- Confirmar la transacción
    COMMIT TRANSACTION;
    PRINT 'Transacción completada con éxito';
END TRY
BEGIN CATCH
    -- Revertir la transacción en caso de error
    ROLLBACK TRANSACTION;
    PRINT 'Error en la transacción. Operación revertida.';
    DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
    PRINT @ErrorMessage;
END CATCH;
```

Messages

(1 row affected)  
Error en la transacción. Operación revertida.  
Error simulado después de insertar Pais

### Resultado obtenido:

- En el caso de haber algún tipo de error en la ejecución de la transacción el CATCH atrapa el error, ejecutando a su vez el ROLLBACK TRANSACTION, el cual revierte toda operación realizada, evitando que se guarden los registros corrompidos en la base de datos.

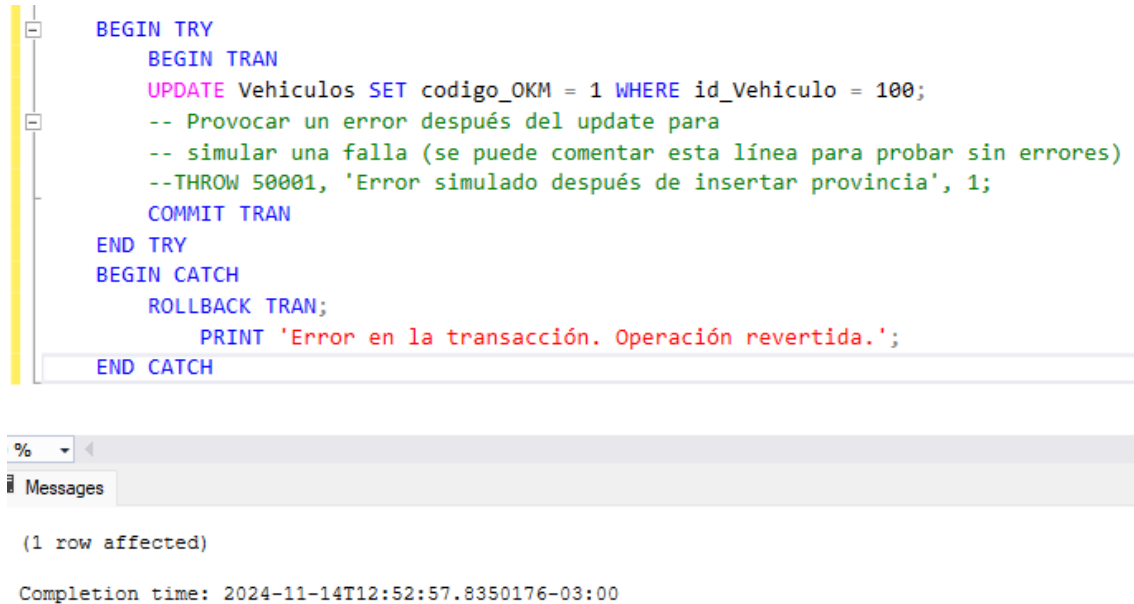
### Ejemplo práctico aplicado para actualizar datos:

```
BEGIN TRY
    BEGIN TRAN
    UPDATE Vehiculos SET codigo_OKM = 1 WHERE id_Vehiculo = 100;
    -- Provocar un error después del update para
    -- simular una falla (se puede comentar esta línea para probar sin errores)
    --THROW 50001, 'Error simulado después de insertar provincia', 1;
    COMMIT TRAN
END TRY
BEGIN CATCH
```

```
ROLLBACK TRAN;

PRINT 'Error en la transacción. Operación revertida.';

END CATCH
```



The screenshot shows a SQL Server Enterprise Manager window with a T-SQL script editor and a Messages pane. The script is a TRY-CATCH block that attempts to update the 'codigo\_OKM' column in the 'Vehiculos' table for 'id\_Vehiculo = 100'. It includes comments in Spanish and a simulated error using 'THROW 50001'. The Messages pane shows that 1 row was affected and provides the completion time: 2024-11-14T12:52:57.8350176-03:00.

```
BEGIN TRY
    BEGIN TRAN
    UPDATE Vehiculos SET codigo_OKM = 1 WHERE id_Vehiculo = 100;
    -- Provocar un error después del update para
    -- simular una falla (se puede comentar esta línea para probar sin errores)
    --THROW 50001, 'Error simulado después de insertar provincia', 1;
    COMMIT TRAN
END TRY
BEGIN CATCH
    ROLLBACK TRAN;
    PRINT 'Error en la transacción. Operación revertida.';
END CATCH
```

Messages

(1 row affected)

Completion time: 2024-11-14T12:52:57.8350176-03:00

### Resultado obtenido:

- Se logró aplicar los cambios en la tabla Vehiculos del registro en particular al cual se desea realizar una actualización de datos .

### Ejemplo práctico aplicado para actualizar datos simulando un error:

```
BEGIN TRY

    BEGIN TRAN

    UPDATE Vehiculos SET codigo_OKM = 1 WHERE id_Vehiculo = 100;

    -- Provocar un error después del update para
    -- simular una falla (se puede comentar esta línea para probar sin errores)
    THROW 50001, 'Error simulado después de insertar provincia', 1;

    COMMIT TRAN

END TRY

BEGIN CATCH

    ROLLBACK TRAN;

    PRINT 'Error en la transacción. Operación revertida.';
```

END CATCH

```
END TRY
BEGIN CATCH
    -- En caso de error, se revierte toda la transacción
    ROLLBACK TRANSACTION;
    PRINT 'Error en la transacción. Operación revertida.';
END CATCH;

BEGIN TRY
    BEGIN TRAN
        UPDATE Vehiculos SET codigo_OKM = 1 WHERE id_Vehiculo = 100;
        -- Provocar un error después del update para
        -- simular una falla (se puede comentar esta línea para probar sin errores)
        THROW 50001, 'Error simulado después de insertar provincia', 1;
    COMMIT TRAN
END TRY
BEGIN CATCH
    ROLLBACK TRAN;
    PRINT 'Error en la transacción. Operación revertida.';
END CATCH
```

%

Messages

(1 row affected)  
Error en la transacción. Operación revertida.

Completion time: 2024-11-14T12:58:56.0899630-03:00

### Resultado obtenido:

- En el caso de la actualización de datos con simulación de un error podemos observar que se ejecuta el **ROLLBACK TRAN** que se encuentra dentro del **CATCH** evitando que los cambios se apliquen al registro esperado.



## Capítulo V: Conclusiones

**Manejo de transacciones y transacciones anidadas:** El manejo de permisos en SQL Server es una herramienta poderosa para proteger la integridad y confidencialidad de los datos. Implementar permisos adecuados según las necesidades garantiza que cada usuario solo pueda realizar las acciones necesarias para su rol, evitando riesgos innecesarios mejora la eficiencia operativa.

**Procedimientos y funciones almacenadas:** Transacciones y transacciones anidadas: Las transacciones en SQL Server son fundamentales para garantizar la integridad, consistencia y confiabilidad de los datos en sistemas de bases de datos. Mediante el uso de transacciones, es posible agrupar una serie de operaciones de forma que se ejecuten en su totalidad o se reviertan en caso de error, cumpliendo con los principios de atomicidad y consistencia. Este enfoque minimiza el riesgo de datos incompletos o inconsistentes, proporcionando una estructura clara para el control de errores y la recuperación.

**Optimización de Consultas a través de Índices en SQL Server:** Implementar índices en SQL Server puede mejorar significativamente el rendimiento de las consultas en tablas grandes. Sin embargo, el tipo de índice seleccionado y su configuración deben estar alineados con los requisitos de consulta y las necesidades de mantenimiento de la base de datos. Un índice agrupado es ideal para ordenar datos en consultas de rangos, mientras que un índice no agrupado con columnas incluidas puede ser útil en situaciones de consultas con múltiples columnas.

**Optimización de Consultas a través de Índices en SQL Server:** Implementar índices en SQL Server puede mejorar significativamente el rendimiento de las consultas en tablas grandes. Sin embargo, el tipo de índice seleccionado y su configuración deben estar alineados con los requisitos de consulta y las necesidades de mantenimiento de la base de datos. Un índice agrupado es ideal para ordenar datos en consultas de rangos, mientras que un índice no agrupado con columnas incluidas puede ser útil en situaciones de consultas con múltiples columnas.

## Capítulo VI. Bibliografía.

[1] Microsoft. "SQL Server Documentation". 2023. [Online]. Available: <https://learn.microsoft.com/es-es/sql/sql-server/?view=sql-server-ver16>