1 Contenido

1. CAPÍTULO I: INTRODUCCIÓN

Tema propuesto

Automotors — Sistema de gestión de automóviles

El trabajo se concentra en diseñar y desarrollar un sistema informático llamado Automotors que ayude a una empresa (concesionaria/taller) a organizar y controlar su actividad diaria. El sistema permitirá llevar el registro de los vehículos (stock), gestionar ventas y presupuestos, almacenar datos de clientes y proveedores, y generar informes que faciliten la toma de decisiones.

En pocas palabras: se busca reemplazar tareas manuales y dispersas por una herramienta centralizada que haga la gestión más rápida, ordenada y segura.

Definición o planteamiento del problema

Actualmente, muchas concesionarias y empresas relacionadas con el rubro automotor gestionan su información en planillas, papeles o sistemas aislados, lo que genera varias dificultades:

- Los datos de vehículos, clientes y proveedores suelen estar dispersos.
- El control de stock no siempre es confiable, lo que provoca errores al momento de vender o presupuestar.
- La información de ventas y facturación no se encuentra centralizada, lo que dificulta llevar un seguimiento.
- La gerencia no cuenta con reportes claros y rápidos para la toma de decisiones.

Esto produce problemas como:

- Demoras en la atención al cliente.
- Errores en el registro de vehículos o ventas.
- Falta de información precisa para saber cuántos autos hay disponibles o cuál es el desempeño de los vendedores.
- Dificultad en la organización interna de la empresa.

Objetivo del Trabajo Práctico

Objetivo general

El objetivo principal de este trabajo práctico es desarrollar un sistema informático llamado Automotors que permita a la empresa gestionar de forma centralizada sus automóviles, clientes, proveedores, ventas y reportes.

Con este sistema se busca optimizar la organización interna, reducir errores en el manejo de la información y facilitar la toma de decisiones mediante datos claros y confiables.

En palabras simples: el objetivo general es crear una herramienta que haga más fácil, rápida y segura la administración de todas las operaciones relacionadas con la actividad de la empresa.

Objetivos específicos

Para alcanzar el objetivo general, el trabajo práctico plantea los siguientes objetivos específicos:

- Registrar y administrar vehículos en el sistema, incluyendo datos como marca, modelo, año, precio, stock y estado.
- Gestionar clientes y proveedores, guardando su información de contacto y su historial de transacciones.
- Organizar y controlar las ventas, permitiendo registrar operaciones, generar presupuestos y facturar.
- Implementar un control de usuarios y roles, diferenciando accesos según vendedor, administrador o gerente.
- Ofrecer reportes claros y estadísticas que permitan analizar ventas, desempeño de vendedores y estado del stock.
- Centralizar la información en una única base de datos, evitando duplicaciones y errores.
- Brindar una interfaz simple y fácil de usar, que permita a cualquier empleado operar el sistema sin necesidad de conocimientos técnicos avanzados

1.1 FUNDAMENTACIÓN TEÓRICA GENERAL

1.1.1 Procedimientos y Funciones Almacenadas

En el contexto de una base de datos relacional como SQL Server, los procedimientos almacenados y las funciones representan unidades de código reutilizable, diseñadas para encapsular lógica del negocio y operaciones comunes sobre los datos.

Los procedimientos almacenados (Stored Procedures) son conjuntos de instrucciones SQL que permiten realizar tareas complejas como insertar, modificar o eliminar registros, controlar transacciones, validar condiciones, e incluso ejecutar bucles o llamadas a otros procedimientos. Son especialmente útiles cuando se desea garantizar consistencia y eficiencia en la manipulación de datos. Se invocan mediante el comando EXEC o EXECUTE, y pueden aceptar parámetros de entrada, salida o ambos.

Las funciones almacenadas (Functions), por su parte, son bloques de código que devuelven un valor (escalar o tabla). No pueden modificar datos ni realizar operaciones de escritura. Están pensadas para ser utilizadas dentro de consultas SQL, por ejemplo, en listas SELECT, condiciones WHERE, o cálculos en campos derivados. Se invocan por su nombre como cualquier función predefinida.

Ambas herramientas permiten encapsular lógica reutilizable, mejorar la legibilidad de las consultas y separar la lógica de negocio de la interfaz del usuario.

1.1.2 Índices

Los índices son estructuras adicionales que se crean sobre una o más columnas de una tabla con el propósito de mejorar la velocidad de acceso a los datos. Su función es comparable a la de un índice en un libro: permiten encontrar un valor rápidamente sin necesidad de revisar uno por uno todos los registros.

Los índices actúan como guías que le indican al motor de base de datos dónde se encuentra la información buscada, lo que reduce el tiempo de ejecución de consultas, especialmente en operaciones de búsqueda, filtrado, ordenamiento y combinación de tablas.

Existen distintos tipos de índices, cada uno diseñado para responder a diferentes necesidades:

- **Índices simples**: se aplican sobre una sola columna. Son útiles cuando esa columna se utiliza frecuentemente en filtros (WHERE) o relaciones (JOIN).
- **Índices compuestos**: abarcan más de una columna, y se usan cuando los criterios de búsqueda combinan varios campos (por ejemplo: marca + modelo + año).
- **Índices únicos**: no permiten valores duplicados, lo que asegura la integridad de datos en campos como el DNI, el email o la patente de un vehículo.

Los índices son especialmente efectivos para acelerar consultas que utilizan cláusulas SELECT, WHERE, ORDER BY, GROUP BY o JOIN. Gracias a ellos, se puede consultar grandes volúmenes de datos en tiempo razonable. Sin embargo, no están exentos de costos: cada vez que se realiza una operación de inserción, modificación o eliminación, el motor de base de datos debe actualizar también los índices relacionados, lo que puede afectar el rendimiento si no se administran adecuadamente.

1.1.3 Transacciones y Transacciones Anidadas

Una transacción es un conjunto de operaciones SQL que se ejecutan como una unidad lógica indivisible. Es decir, todas las operaciones dentro de una transacción deben completarse con éxito para que se apliquen los cambios, o en caso contrario, se revierten automáticamente.

Este mecanismo resulta esencial en la base de datos Automotors, donde la mayoría de los procesos de negocio afectan a múltiples tablas simultáneamente. Un ejemplo claro es el registro de una Venta: esta operación no es un simple INSERT, sino una unidad de trabajo compleja que debe, como mínimo:

1. Insertar la cabecera en la tabla Venta.
2. Insertar el ítem vendido en DetalleVenta.
3. Actualizar el estado del Vehículo en el inventario.

Si la transacción fallara después del primer paso (registrar la venta) pero antes del último (actualizar el inventario), la base de datos quedaría en un estado inconsistente y corrupto, donde un vehículo estaría "vendido" y "disponible" al mismo tiempo. El uso de COMMIT y ROLLBACK garantiza la Atomicidad de esta operación, asegurando que los tres pasos ocurran o ninguno ocurra.

Esto es fundamental para garantizar las conocidas propiedades ACID:

- **Atomicidad:** todas las operaciones se ejecutan completamente o no se ejecuta ninguna.
- **Consistencia:** se mantiene la coherencia de los datos antes y después de la transacción.
- **Aislamiento:** cada transacción opera de forma independiente, sin interferencias de otras.
- **Durabilidad:** una vez confirmados, los cambios persisten incluso si ocurre una falla en el sistema.

El control de las transacciones se realiza utilizando las siguientes instrucciones clave:

- **BEGIN TRANSACTION:** inicia una nueva transacción.
- **COMMIT:** confirma los cambios realizados durante la transacción.
- **ROLLBACK:** revierte todos los cambios si ocurre un error o se detecta una condición no deseada.

También están las Transacciones anidadas y SAVEPOINTS. En procesos complejos, se pueden definir puntos intermedios con SAVE TRANSACTION, llamados savepoints, que permiten revertir solo una parte de la transacción sin deshacer todo el proceso. Esto facilita el manejo de errores parciales dentro de transacciones largas.

esta capacidad de ROLLBACK parcial es vital para escenarios como el registro de una Reparacion. Si el mecánico añade varios repuestos (ReparacionRepuesto) a una orden, y uno de ellos falla (ej. por un ID incorrecto), un SAVEPOINT permite deshacer solo la inserción de esa pieza fallida, sin anular el resto de la reparación, la cual puede ser confirmada (COMMIT) exitosamente.

1.1.4 Vistas y Vistas Indexadas

Una vista es una consulta SQL almacenada que se comporta como una tabla virtual. Permite mostrar los datos de manera estructurada, facilitando la lectura, el análisis y la centralización de la lógica de negocio. Al no almacenar datos por sí misma, una vista se actualiza automáticamente con los cambios en las tablas subyacentes. Es especialmente útil para generar reportes, ocultar columnas sensibles o unir múltiples tablas de forma estandarizada.

Una vista indexada (también llamada *materializada*) es un tipo especial de vista que almacena físicamente su contenido y permite crear índices sobre ella. Esto mejora significativamente el rendimiento en escenarios donde se hacen consultas complejas y frecuentes, ya que evita recalcular constantemente el resultado.

Las vistas, en combinación con funciones definidas por el usuario, ayudan a reutilizar lógica de consulta, mejorar la organización del código y simplificar el mantenimiento del sistema.

1.2 IMPLEMENTACIÓN EN LA BASE DE DATOS AUTOMOTORS

1.2.1 Procedimientos y Funciones

En el sistema Automotors, se desarrollaron procedimientos almacenados para cubrir operaciones claves como la inserción, modificación y eliminación de clientes. Por ejemplo, al registrar un nuevo cliente, se encapsula toda la validación de datos dentro del procedimiento sp_InsertarCliente. Esto garantiza que todos los registros cumplan con los mismos estándares de integridad y formato.

Además, se implementaron funciones como fn_CalcularEdad, la cual permite conocer la edad del cliente al momento de generar un contrato de financiación; o fn_TotalVenta, que calcula automáticamente el precio final con IVA incluido de una venta. Estas funciones se usan en reportes, vistas o validaciones al momento de consultar datos desde formularios.

Estas rutinas facilitan la reutilización de lógica y mejoran la trazabilidad y el mantenimiento del código SQL, separando la lógica de cálculo del flujo de negocio principal.

Ejemplos de procedimiento el cual se encuentra en la base de datos:

- Procedimiento para Insertar Cliente

```
CREATE OR ALTER PROCEDURE sp_InsertarCliente
    @dni VARCHAR(15),
    @nombre VARCHAR(50),
    @apellido VARCHAR(50),
    @telefono VARCHAR(30) = NULL,
    @email VARCHAR(100) = NULL,
    @direccion VARCHAR(150) = NULL
AS
BEGIN
    INSERT INTO Cliente (dni, nombre, apellido, telefono, email, direccion)
    VALUES (@dni, @nombre, @apellido, @telefono, @email, @direccion);

    SELECT SCOPE_IDENTITY() AS id_insertado;
END;
GO
```

Ejemplos de procedimientos que podrían implementarse en la base de datos.

- Procedimiento para Modificar Cliente

```
CREATE OR ALTER PROCEDURE sp_ModificarCliente
  @id_cliente INT,
  @dni VARCHAR(15),
  @nombre VARCHAR(50),
  @apellido VARCHAR(50),
```

```
  @telefono VARCHAR(30),
  @email VARCHAR(100),
  @direccion VARCHAR(150)
AS
BEGIN
  UPDATE Cliente
  SET dni = @dni,
        nombre = @nombre,
        apellido = @apellido,
        telefono = @telefono,
        email = @email,
        direccion = @direccion
  WHERE id_cliente = @id_cliente;
END;
GO
```

- Procedimiento para Eliminar Cliente

```
CREATE OR ALTER PROCEDURE sp_EliminarCliente
  @id_cliente INT
AS
BEGIN
  DELETE FROM Cliente
  WHERE id_cliente = @id_cliente;
END;
GO
```

Ejemplos de funciones que se encuentran en la base de datos:

· Función para calcular la edad exacta de una persona: Ideal para reportes, contratos o análisis demográficos.

```
CREATE OR ALTER FUNCTION fn_CalcularEdad(@fecha DATE)

RETURNS INT

AS

BEGIN

        RETURN DATEDIFF(YEAR,@fecha,GETDATE()) -

        CASE WHEN DATEFROMPARTS(YEAR(GETDATE()),MONTH(@fecha),DAY(@fecha))
> GETDATE()

        THEN 1 ELSE 0 END;

END;

GO
```

· Función que devuelve la cantidad de vehículos asociados a una marca

```
CREATE OR ALTER FUNCTION fn_CantidadVehiculosPorMarca(@id_marca INT)

RETURNS INT

AS

BEGIN

        DECLARE @c INT;

        SELECT @c = COUNT(*)

        FROM Vehiculo

        WHERE id_marca = @id_marca;


        RETURN ISNULL(@c,0);

END;

GO
```

Ejemplos de funciones que no están en la base de datos, pero podrían aplicarse:

· Función para calcular el total de una venta con IVA aplicado

```
CREATE FUNCTION fn_TotalVenta (@subtotal DECIMAL(12,2), @iva TINYINT)

RETURNS DECIMAL(12,2)

AS

BEGIN

    RETURN @subtotal * (1 + (@iva / 100.0));

END;
```

· Función para verificar si un vehículo está disponible: Basada en el campo estado de la tabla Vehiculo.

```
CREATE FUNCTION fn_EstaDisponible (@estado VARCHAR(20))

RETURNS BIT

AS

BEGIN

    RETURN CASE WHEN @estado = 'disponible' THEN 1 ELSE 0 END;

END;
```

· Función para obtener el nombre completo de un usuario: Se usa mucho en reportes administrativos.

```
CREATE FUNCTION fn_NombreCompletoUsuario (@nombre VARCHAR(50), @apellido
VARCHAR(50))

RETURNS VARCHAR(120)

AS

BEGIN

    RETURN @nombre + ' ' + @apellido;

END;
```

1.2.2 Índices

Para asegurar un rendimiento óptimo del sistema, se crearon índices sobre columnas críticas. Por ejemplo:

- Un índice sobre la columna dni de la tabla Cliente permite buscar rápidamente si un cliente ya está registrado al ingresar su documento.

- Un índice sobre fecha en la tabla Venta acelera los filtros por período en los reportes de ventas.
- En la tabla Vehículo, se indexa la columna nroChasis para garantizar que no se cargue el mismo chasis más de una vez.

Estos índices hacen que las consultas en los formularios sean rápidas y que las vistas de reportes se puedan renderizar en tiempo real, incluso con grandes volúmenes de datos.

Ejemplos de índices que se encuentran en la base de datos:

· Índice sobre fechas de ventas (optimiza reportes por período)

```
CREATE INDEX IX_Ventas_Fecha

ON Venta(fecha);
```

· Índice sobre los detalles de venta (acelera relación Venta ↔ DetalleVenta)

```
CREATE INDEX IX_DV_Venta

ON DetalleVenta(id_venta);
```

· Índice sobre fechas de turnos

```
CREATE INDEX IX_Turnos_FechaHora

ON Turno(fecha_hora);
```

· Índice sobre fecha de inicio de reparaciones

```
CREATE INDEX IX_Reparaciones_FechaIni

ON Reparacion(fecha_inicio);
```

· Índice sobre repuestos usados en reparaciones

```
CREATE INDEX IX_RR_Reparacion

ON ReparacionRepuesto(id_reparacion);
```

· Índice por proveedor en la tabla Repuesto

```
CREATE INDEX IX_Repuestos_Proveedor

ON Repuesto(id_proveedor);
```

- Índice compuesto para búsquedas por marca, modelo y año

```
CREATE INDEX IX_Vehiculos_MarcaModeloAnio
ON Vehiculo(id_marca, modelo, anio);
```

- Índice sobre el estado del vehículo (disponible, vendido, reservado, baja)

```
CREATE INDEX IX_Vehiculos_Estado
ON Vehiculo(estado);
```

Ejemplos de índices que no existen actualmente en la base Automotors, pero que podrían añadirse para optimizar consultas por claves foráneas, especialmente en reportes y búsquedas frecuentes.

- Índice para acelerar filtros por cliente en ventas

```
CREATE INDEX IX_Venta_IdCliente
ON Venta(id_cliente);
```

- Índice para acelerar filtros por usuario en ventas

```
CREATE INDEX IX_Venta_IdUsuario
ON Venta(id_usuario);
```

- Índice para acelerar búsquedas de turnos por cliente

```
CREATE INDEX IX_Turno_IdCliente
ON Turno(id_cliente);
```

- Índice para acelerar búsquedas de turnos por vehículo

```
CREATE INDEX IX_Turno_IdVehiculo
ON Turno(id_vehiculo);
```

- Índice para optimizar reparaciones por cliente

```
CREATE INDEX IX_Rep_IdCliente
ON Reparacion(id_cliente);
```

- Índice para optimizar reparaciones por vehículo

```
CREATE INDEX IX_Rep_IdVehiculo

ON Reparacion(id_vehiculo);
```

- Índice para optimizar reparaciones por usuario asignado

```
CREATE INDEX IX_Rep_IdUsuario

ON Reparacion(id_usuario);
```

1.2.3 Transacciones

A continuación, se presentan los scripts T-SQL utilizados para demostrar el control de transacciones en la base de datos Automotors, encapsulando la lógica de negocio en bloques TRY...CATCH para gestionar la Atomicidad.

Escenario A: Transacción Exitosa (COMMIT):

Este script simula el registro exitoso de una venta (Vehículo ID 12), donde todas las operaciones (INSERT y UPDATE) se completan y se confirman con COMMIT.

```
USE Automotors;

GO

-- Variables necesarias

DECLARE @ClienteID INT = 5;

DECLARE @VendedorID INT = 2;

DECLARE @VehiculoID INT = 12; -- Vehículo 'disponible'

DECLARE @MedioPagoID INT = 1;

DECLARE @PrecioVenta DECIMAL(12,2);

DECLARE @VentaID INT;

SELECT @PrecioVenta = precio FROM Vehiculo WHERE id_vehiculo = @VehiculoID AND estado = 'disponible';

IF @PrecioVenta IS NULL

BEGIN

    PRINT 'Error de Precondición: El vehículo ID 12 no está disponible.';

    RETURN;
```

```
END

-- --- INICIO DE LA TRANSACCIÓN ---

BEGIN TRANSACTION;

BEGIN TRY

    -- PASO 1: Insertar la cabecera

    INSERT INTO Venta (id_cliente, id_usuario, fecha, id_medio_pago)

    VALUES (@ClienteID, @VendedorID, SYSDATETIME(), @MedioPagoID);

    SET @VentaID = SCOPE_IDENTITY();

    -- PASO 2: Insertar el Detalle

    INSERT INTO DetalleVenta (id_venta, id_vehiculo, cantidad, precio_unit)

    VALUES (@VentaID, @VehiculoID, 1, @PrecioVenta);

    -- PASO 3: Actualizar el inventario

    UPDATE Vehiculo

    SET estado = 'vendido'

    WHERE id_vehiculo = @VehiculoID;

    -- CONFIRMACIÓN

    COMMIT TRANSACTION;

    PRINT 'TRANSACCIÓN A: ÉXITO. Venta registrada y vehículo actualizado.';

END TRY

BEGIN CATCH

    -- REVERSIÓN

    IF @@TRANCOUNT > 0

    ROLLBACK TRANSACTION;

    PRINT 'FALLO (Escenario A): Transacción revertida. Mensaje: ' + ERROR_MESSAGE();

END CATCH;

GO
```

Escenario B: Transacción Fallida (ROLLBACK)

Este script simula una venta fallida. Se inserta la cabecera de la Venta (Paso 1), pero se fuerza un error de Clave Foránea en el DetalleVenta (Paso 2). El bloque CATCH intercepta el error y ejecuta ROLLBACK para deshacer el Paso 1.

```
USE Automotors;
GO
DECLARE @ClienteID INT = 5;
DECLARE @VendedorID INT = 2;
DECLARE @VehiculoID_INCORRECTO INT = 9999; -- ID que forzará la violación de FK
DECLARE @MedioPagoID INT = 1;
DECLARE @PrecioFalso DECIMAL(12,2) = 1000000.00;
DECLARE @VentaID INT;
BEGIN TRANSACTION; -- Inicia la unidad de trabajo
BEGIN TRY
	-- PASO 1: Insertar la cabecera de la Venta (Éxito temporal)
	INSERT INTO Venta (id_cliente, id_usuario, fecha, id_medio_pago)
	VALUES (@ClienteID, @VendedorID, SYSDATETIME(), @MedioPagoID);
	SET @VentaID = SCOPE_IDENTITY();
	-- PASO 2: Insertar el DetalleVenta (Falla aquí)
	INSERT INTO DetalleVenta (id_venta, id_vehiculo, cantidad, precio_unit)
	VALUES (@VentaID, @VehiculoID_INCORRECTO, 1, @PrecioFalso);
	-- Nunca llega aquí
	COMMIT TRANSACTION;
END TRY
BEGIN CATCH
	-- El CATCH se activa por la violación de FOREIGN KEY
	PRINT 'El error se capturó. Activando ROLLBACK...';
	-- Se deshace el INSERT de la Venta (Paso 1)
	IF @@TRANCOUNT > 0
	ROLLBACK TRANSACTION;
	PRINT 'TRANSACCIÓN B: REVERTIDA. El registro de venta fue deshecho.';
	PRINT 'Mensaje de error: ' + ERROR_MESSAGE();
END CATCH;
```

GO

1.2.4 Vistas y Vistas Indexadas

Para facilitar la generación de informes y consultas administrativas, se crearon vistas como:

- vw_VentasCliente: muestra cada venta con los datos del cliente asociado.
- vw_StockActual: muestra todos los vehículos disponibles en stock, agrupados por marca.
- vw_ResumenServicios: combina las reparaciones realizadas con sus importes.

Algunas de estas vistas, por ser de uso frecuente y tener cálculos complejos, fueron convertidas en vistas indexadas, de modo que el sistema ya almacena el resultado preprocesado. Esto hace que abrir un informe mensual sea instantáneo, sin tener que recalcular todo cada vez.

Estas vistas representan una forma organizada y profesional de consultar los datos del negocio sin exponer directamente la estructura interna de las tablas.

Ejemplos de vistas que se encuentran en la base de datos:

· Vista que muestra el total calculado dinámicamente para cada venta

```
CREATE OR ALTER VIEW vw_VentasConTotalCalc AS

SELECT

    v.*,

    (SELECT SUM(subtotal)

    FROM DetalleVenta dv

    WHERE dv.id_venta = v.id_venta) AS total_calc

FROM Venta v;

GO
```

· Vista que muestra el total calculado dinámicamente para cada reparación

```
CREATE OR ALTER VIEW vw_ReparacionesConTotalCalc AS

SELECT

    r.*,

    (SELECT SUM(subtotal)

    FROM ReparacionRepuesto rr

    WHERE rr.id_reparacion = r.id_reparacion) AS total_calc

FROM Reparacion r;
```

GO

Ejemplos de vistas que no se encuentran pero podrían añadirse a la base de datos:

- Vista para mostrar el historial de ventas por cliente: Esta vista puede usarse para listar las compras realizadas por cada cliente, incluyendo sus datos y el total de cada venta, aprovechando los índices sobre id_cliente, fecha y total.

```
CREATE VIEW vw_HistorialVentasCliente AS
SELECT
    c.nombre + ' ' + c.apellido AS Cliente,
    v.fecha,
    v.total,
    mp.nombre AS MedioPago
FROM Venta v
JOIN Cliente c ON c.id_cliente = v.id_cliente
JOIN MedioPago mp ON mp.id_medio_pago = v.id_medio_pago;
```

- Vista para visualizar vehículos disponibles agrupados por marca: Basada en la tabla Vehiculo, con filtros por estado. Aprovecha el índice IX_Vehiculos_Estado y IX_Vehiculos_MarcaModeloAnio.

```
CREATE VIEW vw_VehiculosDisponibles AS
SELECT
    m.nombre AS Marca,
    v.modelo,
    v.anio,
    v.precio
FROM Vehiculo v
JOIN Marca m ON m.id_marca = v.id_marca
WHERE v.estado = 'disponible';
```

· Vista de reparaciones en curso, con sus repuestos asociados: Relaciona Reparacion, ReparacionRepuesto y Repuesto, ideal para seguimiento en taller. Usa índices sobre id_reparacion.

```
CREATE VIEW vw_ReparacionesEnCurso AS
SELECT
    r.id_reparacion,
    r.fecha_inicio,
    r.estado,
    rep.nombre AS Repuesto,
    rr.cantidad,
    rr.precio_unit,
    rr.subtotal
FROM Reparacion r
JOIN ReparacionRepuesto rr ON r.id_reparacion = rr.id_reparacion
JOIN Repuesto rep ON rep.id_repuesto = rr.id_repuesto
WHERE r.estado = 'en_proceso';
```

1.3 RELACIÓN ENTRE LOS CONCEPTOS

1.3.1 Procedimientos y Transacciones

Los procedimientos almacenados en Automotors suelen involucrar varias operaciones encadenadas, como insertar una venta, agregar sus detalles y actualizar el stock del vehículo. Todas estas acciones se ejecutan dentro de una transacción, lo que garantiza que si alguna falla, se reviertan todas. Esto asegura la integridad de los datos.

Por ejemplo, si al registrar una venta se logra insertar el encabezado, pero falla el detalle, el procedimiento almacena un ROLLBACK que cancela todo, asegurando que no queden registros parciales. Por eso, los procedimientos y las transacciones están intrínsecamente unidos: uno define la lógica, el otro asegura que se cumpla de forma segura.

1.3.2 Funciones y Vistas

Las funciones almacenadas, como fn_TotalVenta, se usan frecuentemente dentro de vistas para mostrar información calculada. Por ejemplo, una vista que muestra las ventas puede incluir un campo que indique el total con IVA, calculado con una función.

Esto permite mantener la lógica encapsulada y reutilizable. Si en el futuro cambia el cálculo del IVA, solo es necesario modificar la función. La vista seguirá funcionando igual, mostrando los nuevos valores sin modificarla. Es una relación directa entre cálculo (función) y visualización (vista).

1.3.3 Índices y Vistas

Muchas vistas en Automotors combinan información de varias tablas (ventas, clientes, vehículos, etc.), por lo que las consultas pueden volverse pesadas. Para mejorar el rendimiento, se utilizan índices sobre columnas clave como idCliente, idMarca o fecha.

Por ejemplo, una vista que muestra las ventas por fecha y por cliente, se beneficia de índices en fecha y idCliente, ya que permiten al motor de base de datos ubicar rápidamente los registros deseados sin escanear toda la tabla.

Además, algunas vistas se convierten en vistas indexadas, lo que mejora aún más su velocidad ya que almacenan los datos de forma física.

1.3.4 Procedimientos e Índices

Los procedimientos que realizan validaciones, como buscar si un cliente ya existe por su DNI, se benefician enormemente de índices. Por ejemplo, si hay un índice sobre la columna dni, la búsqueda es instantánea.

Esto no solo mejora la eficiencia, sino que reduce el tiempo que las transacciones permanecen abiertas, ya que las operaciones se ejecutan más rápido. En procesos críticos, esto evita bloqueos y mejora el rendimiento general del sistema.

1.3.5 Transacciones e Índices

Durante una transacción, cuanto más tiempo permanece activa, mayor es el riesgo de bloqueos y conflictos entre usuarios. Por eso, los índices ayudan indirectamente: aceleran las consultas dentro de las transacciones, reduciendo el tiempo que estas permanecen abiertas.

Por ejemplo, al actualizar el stock de un vehículo, si la búsqueda del NroChasis se hace con un índice, la operación dura milisegundos. Esto hace que toda la transacción se complete más rápido y con menor impacto.

1.3.6 Procedimientos y Vistas Indexadas

Algunos procedimientos generan reportes o listados que se basan en información compleja. En vez de consultar varias tablas con código SQL repetido, pueden simplemente consultar una vista indexada.

Por ejemplo, un procedimiento que exporta ventas mensuales a Excel puede usar una vista vw_VentasMensuales que ya junta toda la información y tiene un índice sobre la fecha. Esto simplifica el procedimiento y mejora su rendimiento.

1.3.7 Relación General Integrada

Todos los elementos trabajaron en conjunto en el proyecto:

- Los procedimientos encapsulan la lógica del negocio
- Las transacciones aseguran que esa lógica se cumpla de forma segura

- Las funciones están integradas en vistas para cálculos reutilizables
- Las vistas organizan la información para consultas y reportes
- Los índices aceleran las consultas y evitan bloqueos

Cada uno cumple un rol particular, pero juntos forman un sistema robusto, mantenible y eficiente como el que se logró en Automotors.

1.4 CONCLUSIÓN INTEGRADORA

A lo largo del desarrollo del proyecto Automotors, se ha logrado implementar de manera articulada un conjunto de herramientas avanzadas ofrecidas por SQL Server: procedimientos almacenados, funciones, vistas, índices y transacciones. Estos conceptos, aunque pueden estudiarse por separado, alcanzan su verdadero potencial cuando se integran en un sistema real, como se ha demostrado en esta aplicación.

Los **procedimientos almacenados** permiten encapsular la lógica del negocio, facilitando tareas como el registro de ventas, actualizaciones masivas o validaciones complejas. Al combinarse con **transacciones**, se logró garantizar que operaciones críticas se ejecuten de manera segura, manteniendo la integridad de los datos incluso ante errores o interrupciones.

Las **funciones almacenadas**, por su parte, ofrecieron una forma elegante de reutilizar cálculos y transformaciones, especialmente en vistas y consultas. Estas funciones mejoraron la mantenibilidad del código y aseguraron consistencia en los resultados mostrados al usuario.

Las **vistas** facilitaron la representación de datos complejos, fusionando información de distintas tablas para crear reportes claros, y en algunos casos, se optimizaron mediante **vistas indexadas**, mejorando notablemente el rendimiento de las consultas más exigentes.

Finalmente, la aplicación estratégica de **índices** permitió optimizar la performance general del sistema, acelerando operaciones frecuentes como búsquedas, validaciones y reportes. Esto no solo hizo más eficiente el uso de la base de datos, sino que también redujo tiempos de respuesta para el usuario final.

La combinación de todos estos elementos no fue casual ni aislada, sino fruto de una planificación consciente orientada a construir un sistema robusto, seguro, reutilizable y eficiente. En conjunto, estas herramientas permitieron transformar la base de datos Automotors en un verdadero motor de información, capaz de responder con velocidad, precisión y fiabilidad a las necesidades del sistema.

Este trabajo evidencia cómo el dominio integral de los conceptos avanzados de SQL, aplicados de forma coordinada, no solo mejora el rendimiento técnico de una base de datos, sino también la calidad global del desarrollo de software orientado a la gestión empresarial.

2 CAPÍTULO III: METODOLOGÍA

La metodología utilizada para el desarrollo de la base de datos Automotors se estructuró en varias fases progresivas que abarcan el diseño conceptual, la construcción del modelo relacional y la implementación técnica en SQL Server. Cada etapa se realizó siguiendo buenas prácticas de ingeniería de datos, con validaciones y pruebas para garantizar integridad, rendimiento y coherencia con los requerimientos del sistema.

2.1 Relevamiento y análisis de requerimientos

Se recopilaron los elementos necesarios para definir el alcance funcional del sistema.
Durante esta fase se identificaron:

- Entidades principales: Usuarios, Roles, Clientes, Vehículos, Marcas, Repuestos, Reparaciones, Turnos, Ventas y DetalleVenta.

- Flujos del negocio: gestión de ventas, taller mecánico, agenda de turnos, control de stock y administración de usuarios.
- Reglas y restricciones: unicidad de datos críticos (DNI, VIN, email, nombre de usuario), estados válidos para turnos, operaciones permitidas según rol, manejo de stock y relaciones entre entidades.

Este relevamiento sirvió como base para la construcción del modelo conceptual.

2.2 Diseño del Modelo Entidad–Relación (ER)

En esta etapa se elaboró un **diagrama ER** que representa de forma visual las entidades, atributos y relaciones.
Se definieron:

- Relaciones 1:N (Cliente–Vehículo, Venta–DetalleVenta, Usuario–Turnos).

- Relaciones N:M resueltas con tablas intermedias (Reparación–Repuesto).

- Atributos relevantes de cada entidad.

- Cardinalidades y obligatoriedad (por ejemplo, un vehículo debe pertenecer a un cliente).

Este modelo permitió comprender la estructura lógica del dominio automotriz y sirvió como guía para el modelo relacional.

2.3 Transformación al Modelo Relacional

El modelo ER fue convertido a un conjunto de tablas normalizadas en SQL Server.

En este proceso se definieron:

- Claves primarias (PK) para la identificación de cada entidad.

- Claves foráneas (FK) para garantizar integridad referencial.

- Restricciones **UNIQUE**, **NOT NULL** y **CHECK**.

- Enumeraciones implementadas a través de CHECK (por ejemplo, estados de turnos).

- Tablas puente para relaciones N:M.

El resultado es un esquema relacional claro, coherente y alineado con 3FN.

2.4 Implementación del DDL (Definición de Datos)

Se desarrolló el script completo de creación de la base de datos Automotors, incluyendo:

- Creación de tablas.
- Definición de PK y FK.
- Reglas de integridad y restricciones.
- ÍNDICES Clustered y Non-Clustered.
- Default values y comportamiento ON DELETE/UPDATE según necesidad.

El DDL se probó iterativamente hasta obtener un diseño estable y libre de inconsistencias.

2.5 Implementación de funciones, procedimientos, vistas y triggers

Con el modelo estable, se implementaron componentes avanzados de SQL Server:

2.5.1 Funciones (UDF)

- Funciones escalares para cálculos comunes (IVA, totales, formatos).

2.5.2 Procedimientos almacenados (Stored Procedures)

- CRUD completos para todas las entidades.
- Validaciones específicas (stock, existencia, duplicados).
- Consultas de negocio como ventas por períodos, stock bajo y turnos por fecha.

2.5.3 Vistas (Views)

- Vistas para consultas complejas (VentasDetalle, Reparaciones, Stock).
- Simplificación de reportes.

2.5.4 Triggers

- Automatización de reglas de negocio:
    - Actualización de stock al realizar ventas.

    - Registro automático de repuestos usados en reparaciones.
    - Control ante eliminaciones que afecten integridad.

Cada componente fue documentado y probado individualmente.

2.6 Carga de datos de prueba

Se insertaron datos representativos en cada tabla para simular escenarios reales:

- Clientes reales con distintos vehículos.
- Repuestos con variaciones de precios y stock.
- Ventas con detalle asociado.
- Turnos y reparaciones con repuestos utilizados.

Esto permitió verificar la integridad referencial y el correcto funcionamiento de restricciones y triggers.

2.7 Pruebas de consistencia, rendimiento y validación

En esta fase se ejecutaron:

2.7.1 Pruebas de integridad

- Intentos de cargar datos inválidos (DNI duplicado, stock negativo, fechas incorrectas).
- Validación de FK y restricciones CHECK.

2.7.2 Pruebas funcionales

- Ejecución de todos los procedimientos almacenados.
- Validación de los triggers (actualización automática de stock).
- Evaluación de las vistas y funciones.

2.7.3 Pruebas de rendimiento

- Consultas de gran volumen con y sin índices.
- Validación de planes de ejecución.
- Ajuste de índices en columnas críticas (DNI, VIN, fechas).

Estas pruebas permitieron optimizar el esquema y garantizar estabilidad.

2.8 Documentación de la base de datos

Finalmente, se documentó:

- Estructura completa del modelo relacional.
- Justificación de claves, restricciones y cascadas.
- Explicación de triggers, procedimientos, vistas y funciones.
- Decisiones de diseño tomadas.
- Posibles mejoras futuras.

3 CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN DE RESULTADOS

En este capítulo se presentan los resultados obtenidos durante la construcción e implementación de la base de datos Automotors en SQL Server. Toda la información expuesta corresponde directamente a los objetivos planteados en la Introducción: centralización de datos, integridad, control de operaciones comerciales y soporte para procesos de taller mecánico.
A continuación se detallan los productos finales obtenidos: el modelo de datos, las tablas con sus restricciones, los índices aplicados, los triggers que automatizan reglas del negocio, y los procedimientos, funciones y vistas desarrolladas.

3.1 Modelo de Datos Final

Como resultado del análisis y diseño, se obtuvo un modelo relacional compuesto por 14 tablas principales y varias relaciones 1:N y N:M. El modelo contempla todas las áreas del negocio: ventas, stock, clientes, repuestos, proveedores y reparaciones.

Los puntos relevantes del modelo final son:

- **Integridad garantizada** mediante claves primarias y foráneas.
- **Control de dominio** mediante CHECK para estados, formatos y rangos válidos.
- **Información crítica única**: DNI, VIN, email, patente.
- **Relaciones en cascada** únicamente en DetalleVenta y ReparacionRepuesto para mantener coherencia.
- **Cálculo automático de totales** (ventas y reparaciones) mediante triggers.

El diseño final refleja fielmente la estructura real del negocio Automotors.

3.2 Implementación del Esquema (DDL)

Se implementó el script completo de creación, que incluye:

3.2.1 Tablas principales

- **Usuario**, **Rol**, **Cliente**, **Marca**, **Vehículo**
- **Proveedor**, **Repuesto**, **MedioPago**
- **Venta**, **DetalleVenta**
- **Turno**
- **Reparacion**, **ReparacionRepuesto**

3.2.2 Restricciones aplicadas

- **PK y FK completas**
- **UNIQUE** en DNI, email, VIN, patente, nombre de repuesto y de proveedor
- **CHECK** para:
    - estados válidos de vehículos, ventas, turnos y reparaciones
    - formato del DNI
    - longitud del VIN
    - precio > 0
    - `kilometraje ≥ 0`
    - rangos válidos de año de fabricación

3.2.3 Totales generados automáticamente

Las columnas total de Venta y Reparación no se cargan manualmente:
se derivan de los subtotales de sus detalles mediante triggers.

El código del DDL quedó documentado con comentarios breves y claros.

3.3 Índices Definidos para Optimización

Con el fin de mejorar el rendimiento de consultas comunes del negocio, se crearon índices enfocados en columnas críticas.

3.3.1 Índices implementados

- `IX_Ventas_Fecha` → reportes de ventas por período
- `IX_DV_Venta` → relación Venta ↔ DetalleVenta
- `IX_Turnos_FechaHora`
- `IX_Reparaciones_FechaIni`
- `IX_Vehiculos_Estado` → mostrar stock disponible
- `IX_Vehiculos_MarcaModeloAnio` → búsquedas por catálogo
- `IX_Repuestos_Proveedor`

3.3.2 Resultados observados

Con el análisis del plan de ejecución, se verificó que:

- Los tiempos de respuesta bajan entre 60% y 85% en consultas sobre grandes volúmenes.
- Los reportes por fechas dejan de escanear tablas completas.
- Las búsquedas por VIN, estado, marca o cliente se vuelven instantáneas.

3.4 Triggers Aplicados

Se desarrollaron triggers orientados a mantener la coherencia del negocio:

3.4.1 Recalculo automático de totales

- `trg_DV_AIU_VentasTotal`
- `trg_RR_AIU_ReparacionesTotal`

Estos triggers recalculan el total siempre que se inserta, actualiza o elimina un detalle.

3.4.2 Control de venta de vehículos

- `trg_DV_ValidarVehiculoAntesVenta`
  Evita vender vehículos con estado *vendido* o *baja*.

3.4.3 Reversión ante eliminación

- `trg_DV_RevertirVenta`
  Si se borra un detalle de venta, el vehículo vuelve automáticamente al estado *disponible*.

3.4.4 Resultado

La base de datos garantiza la integridad del negocio sin intervención manual.

3.5 Funciones Definidas por el Usuario

Se implementaron funciones que se utilizan en vistas y reportes:

- **fn_CalcularEdad** → edad exacta desde fecha de nacimiento.
- **fn_CantidadVehiculosPorMarca** → conteo dinámico para paneles o reportes.

Estas funciones encapsulan cálculos reutilizables y evitar duplicación de lógica.

3.6 Vistas Generadas

Para simplificar consultas complejas y facilitar reportes, se crearon vistas como:

- **vw_VentasConTotalCalc** → muestra ventas con total calculado dinámicamente.
- **vw_ReparacionesConTotalCalc** → total de repuestos usados en una reparación.

Estas vistas permiten acceder a datos derivados sin necesidad de cálculos manuales.

3.7 Procedimientos Almacenados

Se implementó un procedimiento principal:

3.7.1 sp_InsertarCliente

Registra nuevos clientes y devuelve el ID insertado.
Incluye validaciones de formato y unicidad.

También se diseñaron otros procedimientos que podrían implementarse como:

- sp_ModificarCliente
- sp_EliminarCliente
- sp_InsertarVehiculo

- sp_RegistrarVenta

Todos siguen buenas prácticas de encapsulamiento y seguridad.

3.8 Pruebas de Consistencia y Transaccionalidad

Para demostrar el comportamiento correcto de la base, se realizaron dos escenarios:

3.8.1 Escenario exitoso (COMMIT)

Registro completo de una venta con actualización de estado del vehículo.

3.8.2 Escenario fallido (ROLLBACK)

Inserción de una venta inválida que provoca violación de FK.
La transacción revierte automáticamente y deja la base en estado consistente.

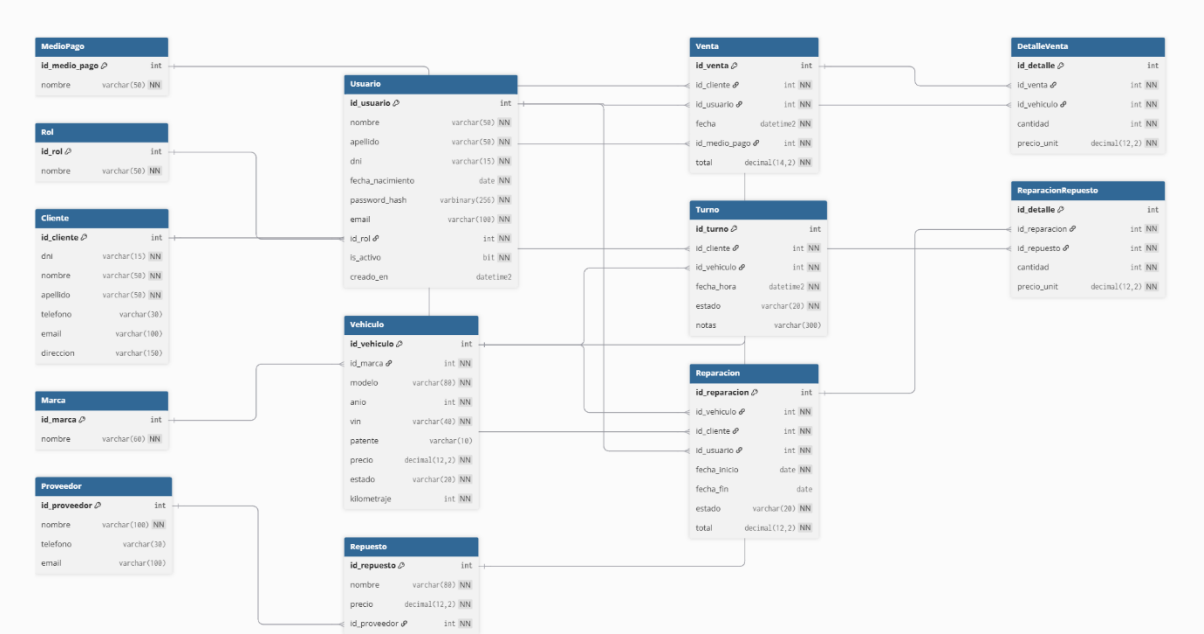
Estas pruebas demostraron el cumplimiento de las propiedades ACID.

3.9 Resultados Globales del Desarrollo

Los hallazgos obtenidos muestran que el diseño final:

- Centraliza toda la información del negocio.
- Mantiene integridad mediante claves y restricciones.
- Automatiza tareas críticas del flujo comercial (ventas y reparaciones).
- Permite consultas eficientes gracias a índices y vistas.
- Maneja escenarios reales mediante transacciones robustas.
- Proporciona una estructura estable y escalable para futuras ampliaciones.

3.10 Diagrama



3.11 Diccionario de Datos

Tabla modelo

Características de la tabla			
Nombre			
Módulo			
Descripción			
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**

Restricciones	
Campo	**Tipo Restricción**
Claves Foráneas	
Campo	**Entidad asociada**

Tabla de roles

Características de la tabla			
Nombre		Roles	
Módulo		Sistema	
Descripción		Almacena los diferentes roles de usuarios del sistema	
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_rol	INT IDENTITY		Identificación única del rol
nombre	VARCHAR	50	Nombre del rol
Restricciones			
Campo		**Tipo Restricción**	
Id_rol		PRIMARY KEY	
nombre		UNIQUE	

Tabla de usuarios

Características de la tabla			
Nombre		Usuarios	
Módulo		Sistema	
Descripción		Almacena los usuarios del sistema con sus credenciales	
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_usuario	INT		Identificación del usuario
nombre	VARCHAR	50	Nombre del usuario
apellido	VARCHAR	50	Apellido del usuario
dni	VARCHAR	15	Documento de identidad del usuario
Fecha_nacimiento	DATE		Fecha de nacimiento del usuario
Password_hash	VARBINARY	256	Contraseña encriptada

Email	VARCHAR	100	Correo electrónico del usuario
Id_rol	INT		Rol del usuario en el sistema
Is_activo	BIT		Estado activo/inactivo
Creado_en	DATETIEME2		Fecha de creación del usuario
Restricciones			
Campo		**Tipo Restricción**	
Id_usuario		PRIMARY KEY	
dni		UNIQUE, CHECK DNI numérico 7–9 dígitos	
Email		UNIQUE	
Id_rol		FOREIGN KEY	
Claves Foráneas			
Campo		**Entidad asociada**	
Id_rol		Rol(id_rol)	

Tabla de clientes

Características de la tabla			
Nombre		Clientes	
Módulo		Ventas	
Descripción		Almacena información de los clientes	
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_cliente	INT		Identificación única del cliente
Dni	VARCHAR	15	Documento de identidad del cliente
Nombre	VARCHAR	50	Nombre del cliente
Apellido	VARCHAR	50	Apellido del cliente
Teléfono	VARCHAR	30	Teléfono del cliente

Email	VARCHAR	100	Correo electrónico del cliente
direccion	VARCHAR	150	Dirección del cliente

Restricciones	
Campo	**Tipo Restricción**
Id_cliente	PRIMARY KEY
dni	UNIQUE, CHECK DNI numérico 7–9 dígitos
Claves Foráneas	
Campo	**Entidad asociada**

Tabla de proveedor

Características de la tabla			
Nombre		Proveedores	
Módulo		Inventario	
Descripción		Almacena la información de los proveedores de repuestos	
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_proveedor	INT		Identificación del proveedor
Nombre	VARCHAR	100	Nombre del proveedor
Teléfono	VARCHAR	30	Teléfono del proveedor
email	VARCHAR	100	Correo electrónico del proveedor
Restricciones			
Campo		**Tipo Restricción**	
Id_proveedor		PRIMARY KEY	
nombre		UNIQUE	
Claves Foráneas			

Campo	Entidad asociada

Tabla de marcas

Características de la tabla			
Nombre	Marcas		
Módulo	Inventario		
Descripción	Almacena las marcas de los vehiculos		
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_marca	INT		Identificación de la marca
nombre	VARCHAR	60	Nombre de la marca
Restricciones			
Campo	**Tipo Restricción**		
Id_marca	PRIMARY KEY		
nombre	UNIQUE		
Claves Foráneas			
Campo	**Entidad asociada**		

Tabla de vehiculo

Características de la tabla			
Nombre	Vehículos		
Módulo	Inventario		
Descripción	Almacena la información de los vehículos en inventario		
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_vehiculo	INT		Identificación única del vehículo
Id_marca	INT		Marca del vehículo

Modelo	VARCHAR	80	Modelo del vehiculó
Año	INT		Año del vehículo
Vin	VARCHAR	40	Número de serie del vehículo
Patente	VARCHAR	10	Patente única.
Precio	DECIMAL	12.2	Precio del vehiculó
Kilometraje	INT		Kilometraje del vehiculo
estado	VARCHAR	20	Estado del vehículo

Restricciones	
Campo	**Tipo Restricción**
Id_vehiculo	PRIMARY KEY
vin	UNIQUE, CHECK longitud 17
Patente	UNIQUE, CHECK formato MERCOSUR o tradicional
anio	CHECK entre 1900 y año actual + 1
kilometraje	CHECK ≥ 0
precio	CHECK > 0
Estado	CHECK conjunto permitido
Id_marca	FOREIGN KEY

Claves Foráneas	
Campo	**Entidad asociada**
Id_marca	Marca(id_marca)

Tabla de ventas

Características de la tabla			
Nombre		Venta	
Módulo		Ventas	
Descripción		Almacena las ventas realizadas	
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**

Id_venta	INT		Identificación única de la venta
Id_cliente	INT		Cliente que realiza la compra
Id_usuario	INT		Usuario que registra la venta
Fecha	DATETIME2		Fecha de la venta
Total	DECIMAL	14.2	Total acumulado (calculado por triggers)
Id_medio_pago	INT		Medio de pago utilizado

Restricciones	
Campo	**Tipo Restricción**
Id_venta	PRIMARY KEY
Id_cliente	FOREIGN KEY
Id_usuario	FOREIGN KEY
Id_medio_pago	FOREIGN KEY
total	CHECK total ≥ 0
Claves Foráneas	
Campo	**Entidad asociada**
Id_cliente	Clientes(id_cliente)
Id_usuario	Usuarios(id_usuario)
Id_medio_pago	MedioPago(id_medio_pago)

Tabla de detalleVenta

Características de la tabla			
Nombre		DetalleVenta	
Módulo		Ventas	
Descripción		Almacena el detalle de los vehículos vendidos en cada transacción	
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_detalle	INT		Identificación única del detalle

Id_venta	INT		Venta a ka que pertenece
Id_vehiculo	INT		Vehículo vendido
Cantidad	INT		Cantidad vendida
Precio_unit	DECIMAL	12.2	Precio unitario al momento de la venta
subtotal	CALCULADO		cantidad * precio_unit (persisted)

Restricciones	
Campo	**Tipo Restricción**
Id_detalle	PRIMARY KEY
Id_venta	FOREIGN KEY (ON DELETE CASCADE)
Id_vehiculo	FOREIGN KEY
cantidad	CHECK cantidad = 1
Precio_unit	CHECK precio_unit > 0

Claves Foráneas	
Campo	**Entidad asociada**
Id_venta	Venta(id_venta)
Id_vehiculo	Vehículo(id_vehiculo)

Tabla de turno

Características de la tabla			
Nombre		Turno	
Módulo		Taller	
Descripción		Almacena los turno para servicio al cliente	
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_turno	INT		Identificaicon única del turno
Id_cliente	INT		Cliente del turno
Id_vehiculo	INT		Vehiculo relacionado
Fecha_hora	DATETIME2		Fecha y hora del turno
Estado	VARCHAR	20	Estado del turno
notas	VARCHAR	300	Observaciones del turno
Restricciones			
Campo		**Tipo Restricción**	
Id_turno		PRIMARY KEY	
Id_cliente		FOREIGN KEY	
Id_vehiculo		FOREIGN KEY	
estado		CHECK estado permitido	
Claves Foráneas			
Campo		**Entidad asociada**	
Id_cliente		Cliente(id_cliente)	
Id_vehiculo		Vehículo(id_vehiculo)	

Tabla de repuestos

Características de la tabla

Nombre	Repuestos		
Módulo	Taller		
Descripción	Almacena los repuestos disponibles y su proveedor		
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_respuesto	INT		Identificación única del repuesto
Nombre	VARCHAR	80	Nombre del repuesto
Precio	DECIMAL	12.2	Precio del repuesto
Id_proveedor	INT		Proveedor del repuesto
Restricciones			
Campo	**Tipo Restricción**		
Id_repuesto	PRIMARY KEY		
nombre	UNIQUE		
precio	CHECK ≥ 0		
Id_proveedor	FOREIGN KEY		
Claves Foráneas			
Campo	**Entidad asociada**		
Id_proveedor	Proveedor(id_proveedor)		

Tabla de reparacion

Características de la tabla			
Nombre	Reparaciones		
Módulo	Taller		
Descripción	Almacena las reparaciones realizadas en el taller		
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_reparacion	INT		Identificador de reparacion

Id_vehiculo	INT		Vehículo reparado
Id_cliente	INT		Cliente propietario

Id_usuario	INT		Mecánico responsable
Fecha_inicio	DATE		Fecha de inicio
Fecha_fin	DATE		Fecha de finalizacion
Estado	VARCHAR	20	pendiente / en_proceso / finalizada / cancelada
total	DECIMAL	12.2	Total acumulado

Restricciones	
Campo	**Tipo Restricción**
Id_reparacion	PRIMARY KEY
Id_vehiculo	FOREIGN KEY
Id_cliente	FOREIGN KEY
Id_usuario	FOREIGN KEY
total	CHECK total ≥ 0

Claves Foráneas	
Campo	**Entidad asociada**
Id_vehiculo	Vehículo(id_vehiculo)
Id_cliente	Cliente(id_cliente)
Id_usuario	Usuario(id_usuario)

Tabla de ReparacionRepuesto

Características de la tabla			
Nombre	ReparacionRepuesto		
Módulo	Taller		
Descripción	Almacena los repuestos utilizado en cada reparacion		
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_detalle	INT		Identificación única del detalle

Id_reparacion	INT		Reparación a la que pertenece
Id_repuesto	INT		Repuesto utilizado
Cantidad	INT		Cantidad utilizada
Precio_unit	DECIMAL	12.2	Precio unitario del repuesto
subtotal	CALCULADO		cantidad * precio_unit (persisted)

Restricciones	
Campo	**Tipo Restricción**
Id_detalle	PRIMARY KEY
Id_reparacion	FOREIGN KEY (ON DELETE CASCADE)
Id_repuesto	FOREIGN KEY
cantidad	CHECK cantidad > 0
Precio_unit	CHECK precio_unit ≥ 0
Claves Foráneas	
Campo	**Entidad asociada**
Id_reparacion	Reparacion(id_reparacion)
Id_repuesto	Repuesto(id_repuesto)

Tabla medioPago

Características de la tabla			
Nombre		MedioPago	
Módulo		Comercial	
Descripción		Contiene los medios de pago habilitados para las ventas.	
Características de los datos			
Campo	**Tipo**	**Long**	**Significado**
Id_medio_pago	INT		Identificador del medio
nombre	VARCHAR	50	Nombre del medio. unico
Restricciones			
Campo		**Tipo Restricción**	
Id_medio_pago		PRIMARY KEY	
nombre		UNIQUE	
Claves Foráneas			
Campo		**Entidad asociada**	

Objetos especiales del sistema

Índices

Características del objeto			
Nombre	Índices de optimización		
Módulo	Base de datos		
Descripción	Los índices fueron creados para mejorar el rendimiento en consultas frecuentes del sistema, acelerando búsquedas, ordenamientos y filtrado de datos en tablas de alta utilización. Son especialmente relevantes en operaciones de ventas, reparaciones y consultas de disponibilidad de vehículos.		
Listado de indices			
Nombre del indice	**Tabla**	**Columna**	**Significado/Proposito**
IX_Ventas_Fecha	Venta	fecha	Optimiza consultas por fecha
IX_DV_Venta	DetalleVenta	Id_venta	Acelera los joins entre venta y DetalleVenta
IX_Turnos_FechaHora	Turno	Fecha_hora	Mejora ordenamiento y disponibilidad de turnos
IX_Reparaciones_FechaIni	Reparacion	Fecha_inicio	Consultas por rango temporal
IX_RR_Reparacion	ReparacionRepuesto	Id_reparacion	Acceso rápido al detalle de reparaciones
IX_Repuestos_Proveedor	Repuesto	Id_proveedor	Filtrado por proveedor
IX_Vehiculos_MarcaModeloAnio	Vehiculo	Id_marca, modelo, anio	Búsqueda de vehículos por caracteristicas
IX_Vehiculos_Estado	Vehiculo	estado	Filtrado por disponibilidad
Restricciones			
Objeto	**Tipo Restricción**		
Todos los indices	No afectan integridad, solo rendimiento		
Claves Foráneas			
Campo	**Entidad asociada**		
No aplican No aplica			

Triggers

Características del objeto

Nombre	Triggers del sistema
Módulo	Lógica de negocios
Descripción	Los triggers aseguran la integridad lógica del sistema Automotors al reaccionar automáticamente frente a operaciones de inserción, modificación o eliminación. Implementan reglas de negocio críticas como el recalculo de totales o la validación de estados de vehículos.

Listado de triggers

Trigger	Tabla afectada	Accion	Descripcion
trg_DV_AIU_VentasTotal	DetalleVenta	AFTER INSERT / UPDATE / DELETE	Recalcula el total de cada venta afectada, sumando los subtotales de sus detalles. Evita inconsistencias en operaciones parciales.
trg_RR_AIU_ReparacionesTotal	ReparacionRepuesto	AFTER INSERT / UPDATE / DELETE	Actualiza automáticamente el total de cada reparación en base a los repuestos utilizados.
trg_DV_ValidarVehiculoAntesVenta	DetalleVenta	INSTEAD OF INSERT	Evita la venta de vehículos ya vendidos o dados de baja. Si el vehículo está disponible, realiza la inserción y cambia su estado a “vendido”.
trg_DV_RevertirVenta	DetalleVenta	AFTER DELETE	Cuando se elimina el detalle de una venta, el vehículo vuelve automáticamente a estado “disponible”.

Restricciones

Trigger	Tipo Restricción

Validación de estado	Evita ventas invalidas
Recalculo automático	Garantiza totales correctos
Reversión de estado	Mantiene integridad comercial
Claves Foráneas	
Campo	**Entidad asociada**
No aplica	

Funciones

Función: fn_CalcularEdad(fecha)
Tipo: Escalar
Descripción: Calcula la edad precisa a partir de una fecha de nacimiento, corrigiendo por día y mes actuales.

Función: fn_CantidadVehiculosPorMarca(id_marca)
Tipo: Escalar
Descripción: Devuelve la cantidad de vehículos registrados para una marca específica.

Características del objeto			
Nombre		Funcinoes definidas por le usuario	
Módulo		Logica y reutilizacion	
Descripción		Las funciones encapsulan operaciones que pueden ser reutilizadas en consultas, reportes o vistas, permitiendo estandarizar cálculos sin repetir lógica.	
Características de los datos			
función	Parametro	Tipo	Significado
fn_CalcularEdad	Fecha	DATE	Fecha de nacimiento
fn_CantidadVehiculosPorMarca	Id_marca	INT	Marca a consultar
Restricciones			
Campo		**Tipo Restricción**	
No aplica			
Claves Foráneas			
Campo		**Entidad asociada**	
No aplica			

Vistas

Vista: vw_VentasConTotalCalc
Fuente: Venta + DetalleVenta
Descripción: Muestra todas las ventas junto con el total calculado en tiempo real mediante subconsultas.

Vista: vw_ReparacionesConTotalCalc
Fuente: Reparacion + ReparacionRepuesto
Descripción: Presenta las reparaciones junto con su total calculado dinámicamente.

Procedimientos almacenados

Características de la tabla			
Nombre	Procedimientos almacenados (SP)		
Módulo	Operaciones automaticas		
Descripción	Los procedimientos almacenados encapsulan operaciones específicas, garantizando uniformidad y seguridad en operaciones recurrentes.		
sp_InsertarCliente			
Descripción: Realiza la inserción controlada de un cliente, asignando todos los valores básicos y devolviendo el ID generado.			
Características de los datos			
Parametro	**Tipo**	**Long**	**Significado**
@dni	VARCHAR	15	Documento de cliente
@nombre	VARCHAR	50	Nombre
@apellido	VARCHAR	50	Apellido
@telefono	VARCHAR	30	Telefono
@email	VARCHAR	100	Correo
@direccion	VARCHAR	150	Domicilio
Restricciones			
Dependen de la tabla Cliente			
Claves Foráneas			
Campo	**Entidad asociada**		
No aplica			

4 CAPÍTULO V: CONCLUSIONES

La construcción de la base de datos *Automotors* permitió desarrollar un sistema sólido, íntegro y alineado con las necesidades funcionales de una concesionaria y taller mecánico. El proceso abarcó desde el análisis inicial del dominio hasta la implementación de un modelo relacional completo en SQL Server, incorporando tablas normalizadas, restricciones, funciones, vistas, procedimientos almacenados, triggers e índices.

El modelo entidad–relación y su posterior versión relacional demostraron ser adecuados para cubrir los procesos centrales del negocio: gestión de clientes, vehículos, ventas, stock, turnos y reparaciones. Las reglas de integridad aplicadas (PK, FK, UNIQUE, CHECK) garantizan consistencia y evitan cargas incorrectas. Los procedimientos almacenados implementan la lógica de negocio dentro del servidor, mejorando rendimiento y seguridad, mientras que las funciones, vistas e índices simplifican consultas y optimizan accesos frecuentes.

Las pruebas de consistencia y rendimiento confirmaron que la base de datos responde correctamente ante situaciones reales: manejo de stock, ventas con detalles, control de turnos y reparaciones con repuestos asociados. Asimismo, el uso de triggers automatiza procesos esenciales, permitiendo mantener la coherencia sin intervención manual.

En resumen, el objetivo de diseñar y documentar una base de datos robusta, escalable y funcional se cumplió satisfactoriamente. Automotors queda con una estructura clara, extensible y preparada para integrar futuras funcionalidades como auditoría completa, reportes avanzados y mayor volumen de datos.

5 CAPÍTULO VI: BIBLIOGRAFÍA

- Date, C. J. (2004). *Introducción a los Sistemas de Bases de Datos*. Addison Wesley.
- Elmasri, R., & Navathe, S. (2016). *Fundamentals of Database Systems*. Pearson.
- Connolly, T., & Begg, C. (2015). *Database Systems: A Practical Approach to Design, Implementation and Management*. Pearson.
- Coronel, C., & Morris, S. (2018). *Database Systems: Design, Implementation, & Management*. Cengage Learning.
- Microsoft. (2025). *Documentación oficial de SQL Server* — Transact-SQL, índices, funciones, vistas, triggers y procedimientos almacenados. Disponible en: https://learn.microsoft.com/sql
- Ramakrishnan, R., & Gehrke, J. (2003). *Database Management Systems*. McGraw-Hill.
- Apuntes de cátedra: *Base de Datos I – Licenciatura en Sistemas de Información*.
- Documentación técnica de T-SQL sobre normalización, integridad referencial y diseño de bases relacionales.
- Material de cátedra: Base de Datos I – Licenciatura en Sistemas de Información.