# REPORTE DE PRÁCTICA NO. 4

## Práctica 0

**ALUMNO:**
Ariana Garcia Melo

# 1. Introducción

La gestión eficiente de flotillas de vehículos es esencial para las empresas que dependen del transporte de bienes y personal. Una base de datos bien estructurada permite controlar aspectos clave como el mantenimiento, consumo de combustible y la documentación de los vehículos. En este reporte, se presentan diversas consultas en álgebra relacional y SQL diseñadas para extraer información relevante de un sistema de gestión de flotillas.

El correcto manejo de la información en la administración de flotillas permite reducir costos operativos, mejorar la planificación de mantenimiento y optimizar la asignación de recursos. Sin un sistema adecuado, la empresa puede enfrentar problemas como el incumplimiento de normativas, el deterioro prematuro de los vehículos y una gestión ineficiente del consumo de combustible.

En el ámbito de la gestión de bases de datos, la optimización y el manejo eficiente de la información son fundamentales para garantizar la integridad, seguridad y rapidez en las operaciones. Las bases de datos modernas implementan diversos mecanismos para mejorar el procesamiento y la administración de los datos almacenados, especialmente en sistemas de información críticos como la gestión de flotillas, donde se requiere un control preciso de los vehículos, conductores, mantenimiento, consumo de combustible y documentación.

Dentro de estos mecanismos, los procedimientos almacenados, funciones, estructuras de control condicionales y repetitivas, y disparadores (triggers) juegan un papel clave en la automatización y optimización de los procesos dentro del sistema de base de datos.

# 2. Marco teórico

Los sistemas de gestión de bases de datos han evolucionado para ofrecer herramientas que permitan automatizar procesos, mejorar la eficiencia en la manipulación de datos y garantizar la integridad de la información. En este contexto, los procedimientos almacenados, funciones, estructuras de control y disparadores juegan un papel fundamental en el manejo de la lógica de negocio directamente en el servidor de bases de datos.

## Procedimientos almacenados (Procedure)

Los procedimientos almacenados son bloques de código SQL que se guardan en el sistema de gestión de bases de datos y pueden ejecutarse varias veces sin necesidad de reescribir la lógica cada vez que se necesiten. Su uso optimiza el rendimiento, ya que la ejecución se realiza en el servidor, reduciendo la carga de la red y mejorando la seguridad al limitar el acceso directo a las tablas de datos. Además, los procedimientos almacenados permiten la parametrización, facilitando la reutilización del código y la modularidad del sistema. Según [4], los procedimientos almacenados son una de las mejores prácticas en bases de datos para mejorar la eficiencia y seguridad en la ejecución de consultas complejas.

## Funciones (Function)

Las funciones en SQL, al igual que los procedimientos almacenados, son fragmentos de código que pueden reutilizarse, pero a diferencia de estos, siempre devuelven un valor. Las funciones pueden emplearse dentro de consultas SQL, lo que permite optimizar la manipulación de datos y mejorar la legibilidad del código. Según [5], las funciones son esenciales para realizar cálculos específicos sobre datos de manera eficiente y estructurada.

## Estructuras de control condicionales y repetitivas

Las estructuras de control condicionales y repetitivas permiten gestionar el flujo de ejecución de los programas dentro de una base de datos. En SQL, estas estructuras incluyen sentencias como IF, CASE, WHILE y LOOP, que facilitan la toma de decisiones y la iteración sobre conjuntos de datos. [3] mencionan que estas estructuras son fundamentales para la implementación de lógica avanzada en bases de datos, permitiendo que las consultas sean más flexibles y adaptativas a diversas condiciones.

## Disparadores (Triggers)

Los disparadores son mecanismos que permiten ejecutar automáticamente una acción en respuesta a eventos como la inserción, actualización o eliminación de datos en una tabla. Su uso es clave para garantizar la integridad de los datos y automatizar procesos sin intervención manual. De acuerdo con Silberschatz, [6], los disparadores son herramientas poderosas para el control de datos, ya que permiten realizar auditorías, validar información y actualizar registros sin la necesidad de una interacción constante del usuario.

## 3. Herramientas empleadas

Para la implementación de las consultas se utilizaron:

- MySQL como gestor de bases de datos.
- MySQL Workbench para la ejecución de consultas y visualización de resultados.
- LaTeX para la elaboración del reporte.
- Respaldo de base de datos: `https://github.com/idkAriana/Bases-de-Datos-Disribuidas/blob/731062213f9611679e6efb60dd197c8d7b5305ce/gestionFlotilla.sql`

# 4. Desarrollo

## Procedimientos almacenados (Procedure)

Ejemplo 1: Registrar un nuevo mantenimiento de vehículo.

```
DELIMITER //
CREATE PROCEDURE RegistrarMantenimiento(
    IN p_vehiculo_id INT,
    IN p_fecha DATE,
    IN p_tipo_mantenimiento VARCHAR(50),
    IN p_descripcion TEXT,
    IN p_costo DECIMAL(10,2),
    IN p_taller VARCHAR(100)
)
BEGIN
    INSERT INTO Mantenimiento (vehiculo_id, fecha, tipo_mantenimiento, descripcion,
        costo, taller)
    VALUES (p_vehiculo_id, p_fecha, p_tipo_mantenimiento, p_descripcion, p_costo,
        p_taller);
END //
DELIMITER ;
```

Listing 1: Script.

```
    CALL RegistrarMantenimiento(1, '2025-03-10', 'Preventivo', 'Cambio de aceite',
        1200, 'Taller X');
```

Listing 2: Ejecutar el procedimiento.

Ejemplo 2: Actualizar el teléfono de un conductor.

```
DELIMITER //
CREATE PROCEDURE ActualizarTelefonoConductor(
    IN p_conductor_id INT,
    IN p_nuevo_telefono VARCHAR(15)
)
BEGIN
    UPDATE Conductor
    SET telefono = p_nuevo_telefono
    WHERE conductor_id = p_conductor_id;
END //
DELIMITER ;
```

Listing 3: Script.

```
    CALL ActualizarTelefonoConductor(3, '5559998888');
```

Listing 4: Ejecutar el procedimiento.

## Funciones (Functions)

Ejemplo 1: Calcular el costo promedio de mantenimiento de un vehículo.

```
DELIMITER //
CREATE FUNCTION CostoPromedioMantenimiento(p_vehiculo_id INT)
RETURNS DECIMAL(10,2) DETERMINISTIC
BEGIN
    DECLARE promedio DECIMAL(10,2);
    SELECT AVG(costo) INTO promedio
    FROM Mantenimiento
    WHERE vehiculo_id = p_vehiculo_id;
    RETURN IFNULL(promedio, 0);
END //
DELIMITER ;
```

Listing 5: Script.

```
SELECT CostoPromedioMantenimiento(1);
```

Listing 6: Uso de la función.

Ejemplo 2: Obtener el número de mantenimientos de un vehículo.

```
DELIMITER //
CREATE FUNCTION ContarMantenimientos(p_vehiculo_id INT)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE cantidad INT;
    SELECT COUNT(*) INTO cantidad
    FROM Mantenimiento
    WHERE vehiculo_id = p_vehiculo_id;
    RETURN cantidad;
END //
DELIMITER ;
```

Listing 7: Script.

```
SELECT ContarMantenimientos(2);
```

Listing 8: Uso de la función.

## Estructuras de Control (Condicionales y Repetitivas)

Ejemplo 1: Uso de IF en un procedimiento para verificar si un vehículo tiene documentación vencida.

```
DELIMITER //
CREATE PROCEDURE VerificarDocumentacionVencida(
    IN p_vehiculo_id INT,
    OUT p_mensaje VARCHAR(100)
)
BEGIN
    DECLARE fecha_actual DATE;
    DECLARE fecha_vencimiento DATE;

    SET fecha_actual = CURDATE();

    SELECT MAX(fecha_vencimiento) INTO fecha_vencimiento
    FROM Documentacion
    WHERE vehiculo_id = p_vehiculo_id;

IF fecha_vencimiento < fecha_actual THEN
    SET p_mensaje = 'El␣vehiculo␣tiene␣documentacion␣vencida.';
ELSE
    SET p_mensaje = 'El␣vehiculo␣tiene␣documentacion␣vigente.';
END IF;
END //
DELIMITER ;
```

Listing 9: Script.

```
CALL VerificarDocumentacionVencida(1, @mensaje);
SELECT @mensaje;
```

Listing 10: Ejecutar el procedimiento.

Ejemplo 2: Uso de WHILE en un procedimiento para listar los conductores.

```
DELIMITER //
CREATE PROCEDURE ListarConductores()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE c_nombre VARCHAR(100);
```

```
    DECLARE cur CURSOR FOR SELECT nombre FROM Conductor;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;

    lectura: LOOP
        FETCH cur INTO c_nombre;
        IF done THEN
            LEAVE lectura;
        END IF;
        SELECT c_nombre;
    END LOOP;

    CLOSE cur;
END //
DELIMITER ;
```

Listing 11: Script.

```
CALL ListarConductores();
```

Listing 12: Ejecutar el procedimiento.

## Disparadores (Triggers)

Ejemplo 1: Disparador para evitar registros de consumo de combustible con cantidad negativa.

```
DELIMITER //
CREATE TRIGGER VerificarConsumoCombustible
BEFORE INSERT ON ConsumoCombustible
FOR EACH ROW
BEGIN
    IF NEW.cantidad_litros <= 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La cantidad de litros debe ser mayor a 0';
    END IF;
END //
DELIMITER ;
```

Listing 13: Script.

```
INSERT INTO ConsumoCombustible (vehiculo_id, fecha, cantidad_litros, costo_total
    )
VALUES (1, '2025-02-20', -10, 500);
```

Listing 14: Prueba.

Ejemplo 2: Disparador para registrar el historial de cambios en el mantenimiento.

```
CREATE TABLE HistorialMantenimiento (
    historial_id INT AUTO_INCREMENT PRIMARY KEY,
    mantenimiento_id INT,
    vehiculo_id INT,
    fecha DATE,
    tipo_mantenimiento VARCHAR(50),
    descripcion TEXT,
    costo DECIMAL(10,2),
    taller VARCHAR(100),
    fecha_modificacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER //
CREATE TRIGGER RegistrarHistorialMantenimiento
BEFORE UPDATE ON Mantenimiento
FOR EACH ROW
BEGIN
```

```
18                  INSERT INTO HistorialMantenimiento (mantenimiento_id, vehiculo_id, fecha,
                        tipo_mantenimiento, descripcion, costo, taller)
19                  VALUES (OLD.mantenimiento_id, OLD.vehiculo_id, OLD.fecha, OLD.
                        tipo_mantenimiento, OLD.descripcion, OLD.costo, OLD.taller);
20              END //
21              DELIMITER ;
```

Listing 15: Script.

```
1           UPDATE Mantenimiento
2            SET costo = 2500
3            WHERE mantenimiento_id = 1;
```

Listing 16: Prueba.

# 5. Conclusiones

El uso de procedimientos almacenados, funciones, estructuras de control condicionales y repetitivas, y disparadores en bases de datos es fundamental para optimizar el rendimiento, garantizar la integridad de los datos y mejorar la eficiencia en la gestión de la información. Estas herramientas permiten trasladar la lógica de negocio al servidor, reduciendo la sobrecarga en las aplicaciones cliente y mejorando la seguridad al restringir el acceso directo a los datos.

Los procedimientos almacenados proporcionan una forma estructurada de ejecutar operaciones repetitivas y complejas de manera eficiente, evitando la redundancia de código y mejorando el mantenimiento del sistema. Por su parte, las funciones permiten realizar cálculos y transformaciones sobre los datos de manera más modular, lo que facilita su integración en consultas SQL.

Las estructuras de control condicionales y repetitivas agregan flexibilidad en la ejecución de código dentro de la base de datos, permitiendo que las operaciones respondan a diferentes condiciones y se realicen iteraciones sobre conjuntos de datos de manera controlada. Mientras tanto, los disparadores ofrecen automatización en la gestión de datos, asegurando la consistencia e integridad de la información sin necesidad de intervención manual.

En el contexto de la base de datos de gestión de flotillas, estas herramientas juegan un papel clave en la administración de vehículos, conductores, consumos de combustible y mantenimiento. Implementarlas adecuadamente permite mejorar la trazabilidad de la información, reducir errores operativos y facilitar la toma de decisiones basada en datos actualizados y confiables.

# Referencias Bibliográficas

# References


[1] Silberschatz, A., Korth, H. F., Sudarshan, S. **(2011)**. Database System Concepts. McGraw-Hill

[2] Documentación oficial de MySQL: `https://dev.mysql.com/doc/`

[3] Connolly, T., & Begg, C. **(2019)**. *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson.

[4] Date, C. J. **(2021)**. *An Introduction to Database Systems* (8th ed.). Pearson.

[5] Elmasri, R., & Navathe, S.**(2020)**. *Fundamentals of Database Systems* (7th ed.). Pearson.

[6] Silberschatz, A., Korth, H. F., & Sudarshan, S. **(2020)**. *Database System Concepts* (7th ed.). McGraw-Hill.