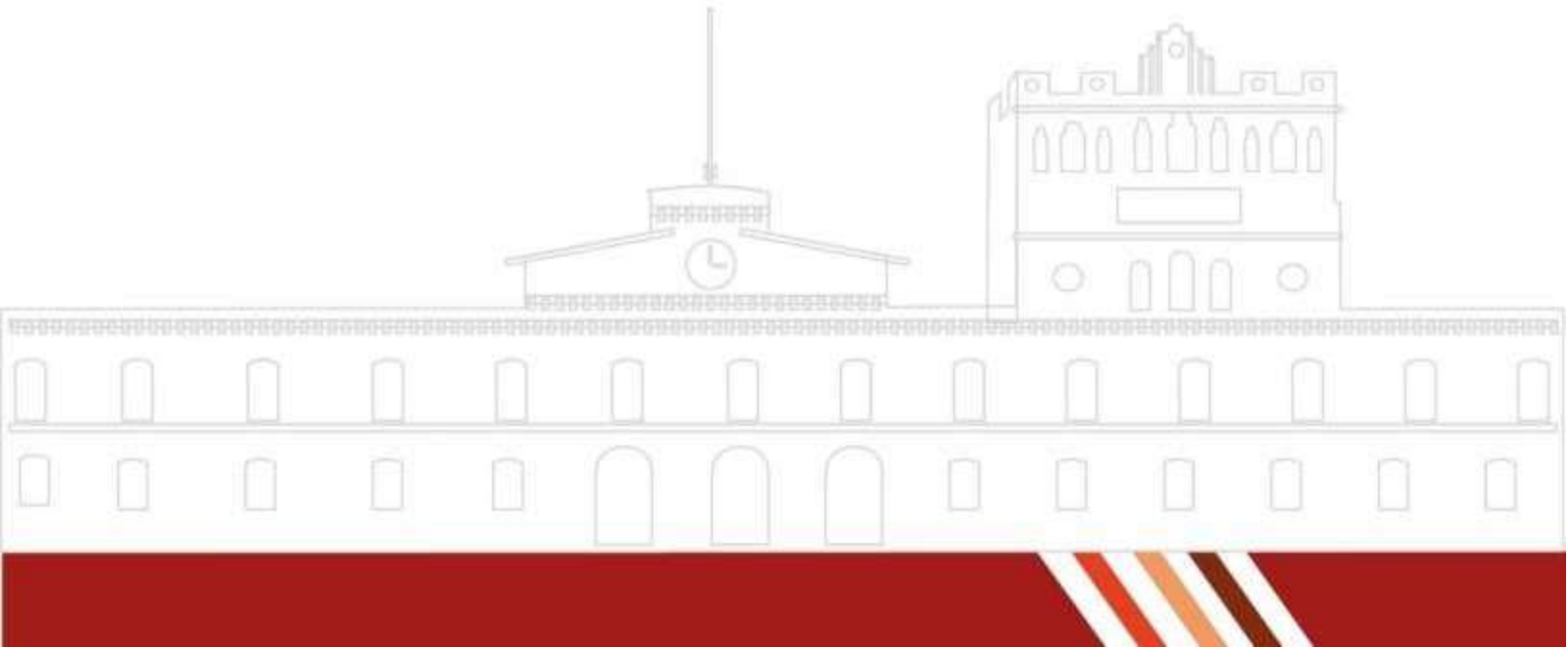
# Reporte de Practica No. 0

## Nombre de la Practica:
## Practica 0

Alumno: Angel Amaya Zumaya

Dr. Eduardo Cornejo Velázquez



## 1. Introducción

Una vez establecida la estructura de una base de datos relacional, el siguiente paso para maximizar su potencial es incorporar lógica de negocio, automatización y seguridad directamente en el motor de la base de datos. Ir más allá de las operaciones básicas (CRUD) y utilizar las capacidades de programación del RDBMS permite crear un sistema más robusto, eficiente y seguro.

Este reporte tiene como objetivo desarrollar el marco teórico y práctico de cuatro conceptos avanzados de SQL: procedimientos almacenados, funciones, estructuras de control y disparadores (triggers). Cada concepto será definido teóricamente y luego ilustrado con dos ejemplos prácticos implementados en MySQL, utilizando el esquema de la base de datos gestion_flotilla previamente diseñada.

El propósito es demostrar cómo estas herramientas pueden encapsular tareas complejas, automatizar procesos de auditoría, validar la integridad de los datos y, en última instancia, reducir la carga en la capa de aplicación, creando un sistema de gestión de flotillas más inteligente y centralizado.

## 2. Marco Teórico

### 2.1. Procedimientos Almacenados (Stored Procedures)

Un procedimiento almacenado es un conjunto de sentencias SQL precompiladas que se almacena en el servidor de la base de datos y puede ser ejecutado con una simple llamada (CALL). Al encapsular lógica de negocio, permiten reutilizar código, reducir el tráfico de red (ya que la lógica se procesa en el servidor) y mejorar la seguridad, al poder otorgar permisos de ejecución sobre el procedimiento sin darlos sobre las tablas subyacentes. Pueden aceptar parámetros de entrada (IN), devolver parámetros de salida (OUT) o ambos (INOUT).

### 2.2. Funciones Almacenadas (Functions)

Una función almacenada es similar a un procedimiento, ya que es un bloque de código reutilizable almacenado en el servidor. Sin embargo, su principal diferencia y característica es que **siempre debe devolver un único valor** (RETURNS). Debido a esto, las funciones pueden ser utilizadas directamente dentro de sentencias SQL (como en un SELECT o WHERE), lo cual es ideal para realizar cálculos o retornar valores derivados que se necesitan en múltiples consultas.

### 2.3. Estructuras de Control

Las estructuras de control son sentencias del lenguaje procedural de SQL que permiten controlar el flujo de ejecución dentro de procedimientos, funciones o triggers. Son la base para añadir lógica condicional y repetitiva.

- **Condicionales (IF, CASE):** Permiten ejecutar diferentes bloques de código basados en la evaluación de una o más condiciones. Son esenciales para la toma de decisiones.
- **Repetitivas (LOOP, WHILE, REPEAT):** Permiten ejecutar un bloque de código múltiples veces hasta que se cumpla una condición de salida. Se utilizan comúnmente para iterar sobre conjuntos de resultados, por ejemplo, con la ayuda de un CURSOR.

2.4. Disparadores (Triggers)

Un disparador, o trigger, es un tipo especial de procedimiento almacenado que no se ejecuta manualmente, sino que se activa **automáticamente** en respuesta a un evento de modificación de datos (INSERT, UPDATE o DELETE) en una tabla específica. Se pueden configurar para que se ejecuten antes (BEFORE) o después (AFTER) del evento. Son una herramienta poderosa para implementar reglas de negocio complejas, crear pistas de auditoría y mantener la integridad referencial de forma automática.

## 3. Desarrollo: Ejemplos de Implementación en MySQL

A continuación, se presentan dos ejemplos prácticos para cada concepto, aplicados al esquema gestion_flotilla.

3.1. Procedimientos Almacenados (Procedure)

*Ejemplo 1: Registrar un Mantenimiento Completo*

Este procedimiento simplifica la tarea de añadir un nuevo registro de mantenimiento, asegurando que se inserten todos los datos necesarios.

```
DELIMITER $$
CREATE PROCEDURE sp_registrar_mantenimiento(
    IN p_idVehiculo INT,
    IN p_idTaller INT,
    IN p_fechaServicio DATE,
    IN p_descripcion TEXT,
    IN p_costo DECIMAL(10, 2),
    IN p_tipoMantenimiento ENUM('Preventivo', 'Correctivo')
)
BEGIN
    INSERT INTO Mantenimiento (idVehiculo, idTaller, fechaServicio, descripcion, costo, tipoMantenimiento)
    VALUES (p_idVehiculo, p_idTaller, p_fechaServicio, p_descripcion, p_costo, p_tipoMantenimiento);
END$$
DELIMITER ;
```

-- Cómo se usaría:
CALL sp_registrar_mantenimiento(1, 1, '2025-10-15', 'Cambio de aceite y filtros', 1500.00, 'Preventivo');

*Ejemplo 2: Obtener Vehículos con Documentos por Vencer*

Este procedimiento es una herramienta administrativa para generar un reporte de vehículos que requieren atención en su documentación en los próximos X días.

```
DELIMITER $$
CREATE PROCEDURE sp_reporte_documentos_por_vencer(
   IN p_dias_limite INT
)
BEGIN
   SELECT
      v.placa,
      v.marca,
      v.modelo,
      d.tipoDocumento,
      d.fechaVencimiento,
      DATEDIFF(d.fechaVencimiento, CURDATE()) AS dias_restantes
   FROM Documento d
   JOIN Vehiculo v ON d.idVehiculo = v.idVehiculo
   WHERE d.fechaVencimiento BETWEEN CURDATE() AND DATE_ADD(CURDATE(),
INTERVAL p_dias_limite DAY)
   ORDER BY dias_restantes ASC;
END$$
DELIMITER ;
```

-- Cómo se usaría (para ver documentos que vencen en los próximos 30 días):
CALL sp_reporte_documentos_por_vencer(30);

3.2. Funciones (Function)

*Ejemplo 1: Calcular Costo Total de Mantenimientos por Vehículo*

Esta función es útil para obtener rápidamente un indicador clave de rendimiento: el costo acumulado de un activo.

```
DELIMITER $$
CREATE FUNCTION fn_costo_total_mantenimiento(
   p_idVehiculo INT
)
RETURNS DECIMAL(12, 2)
DETERMINISTIC
```

```
BEGIN
    DECLARE total DECIMAL(12, 2);

    SELECT SUM(costo) INTO total
    FROM Mantenimiento
    WHERE idVehiculo = p_idVehiculo;

    RETURN IFNULL(total, 0);
END$$
DELIMITER ;

-- Cómo se usaría (directamente en un SELECT):
SELECT placa, marca, modelo, fn_costo_total_mantenimiento(idVehiculo) AS costo_total
FROM Vehiculo;
```

*Ejemplo 2: Verificar Estatus de Licencia de Conductor*

Esta función retorna un estado legible por humanos sobre la vigencia de una licencia, ideal para reportes y validaciones.

```
DELIMITER $$
CREATE FUNCTION fn_estatus_licencia(
    p_idConductor INT
)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE fecha_venc DATE;
    DECLARE estatus VARCHAR(20);

    SELECT fechaVencimientoLicencia INTO fecha_venc
    FROM Conductor
    WHERE idConductor = p_idConductor;

    IF fecha_venc < CURDATE() THEN
        SET estatus = 'Vencida';
    ELSEIF DATEDIFF(fecha_venc, CURDATE()) <= 30 THEN
        SET estatus = 'Próxima a Vencer';
    ELSE
        SET estatus = 'Vigente';
    ENDIF;

    RETURN estatus;
```

```
END$$
DELIMITER ;

-- Cómo se usaría:
SELECT nombre, numeroLicencia, fn_estatus_licencia(idConductor) AS estatus_licencia
FROM Conductor;
```

3.3. Estructuras de Control (Ejemplos de Aplicación)

Las estructuras de control ya fueron utilizadas en los ejemplos anteriores. Aquí se destaca su aplicación.

*Ejemplo 1: Condicional (IF...THEN...ELSEIF)*

La función fn_estatus_licencia es un ejemplo perfecto de una estructura condicional. Evalúa la fecha de vencimiento y, dependiendo del resultado, sigue un camino lógico diferente para asignar el valor de estatus. Esto permite una lógica de negocio compleja dentro de una simple función.

*Ejemplo 2: Repetitiva (CURSOR y LOOP)*

Este procedimiento utiliza un cursor para iterar sobre todos los conductores y una estructura LOOP para realizar una auditoría, registrando en una tabla de logs a aquellos con licencias vencidas.

```
-- Primero, creamos una tabla para el log
CREATE TABLE AuditoriaLicencias (
    idAuditoria INT PRIMARY KEY AUTO_INCREMENT,
    idConductor INT,
    nombreConductor VARCHAR(100),
    fechaVerificacion DATETIME,
    detalle VARCHAR(255)
);

-- Ahora, el procedimiento con el LOOP
DELIMITER $$
CREATE PROCEDURE sp_auditar_licencias_vencidas()
BEGIN
    -- Declaración de variables para almacenar datos del conductor
    DECLARE done INT DEFAULT FALSE;
    DECLARE c_id INT;
    DECLARE c_nombre VARCHAR(100);
    DECLARE c_estatus VARCHAR(20);

    -- Declaración del CURSOR para seleccionar todos los conductores
```

```
    DECLARE cur_conductores CURSOR FOR SELECT idConductor, nombre FROM Conductor;

    -- Declaración del manejador para el final del bucle
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Abrir el cursor
    OPEN cur_conductores;

    -- Iniciar el bucle
    read_loop: LOOP
        -- Obtener la siguiente fila
        FETCH cur_conductores INTO c_id, c_nombre;

        -- Si no hay más filas, salir del bucle
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Usar la función que ya creamos
        SET c_estatus = fn_estatus_licencia(c_id);

        -- Si la licencia está vencida, registrar en la auditoría
        IF c_estatus = 'Vencida' THEN
            INSERT INTO AuditoriaLicencias (idConductor, nombreConductor, fechaVerificacion, detalle)
            VALUES (c_id, c_nombre, NOW(), 'La licencia del conductor se encuentra vencida.');
        END IF;

    END LOOP;

    -- Cerrar el cursor
    CLOSE cur_conductores;
END$$
DELIMITER ;

-- Cómo se usaría para correr la auditoría:
CALL sp_auditar_licencias_vencidas();
```

3.4. Disparadores (Triggers)

*Ejemplo 1: Auditoría de Cambios de Estatus en Rutas*

Este trigger crea un registro histórico cada vez que el estatus de una ruta es modificado, lo cual es vital para auditorías y seguimiento.

```
-- Primero, la tabla para el log
CREATE TABLE LogRutas (
    idLog INT PRIMARY KEY AUTO_INCREMENT,
    idRuta INT,
    fechaCambio DATETIME,
    usuario VARCHAR(100),
    estatusAnterior VARCHAR(50),
    estatusNuevo VARCHAR(50)
);

-- Ahora, el trigger
DELIMITER $$
CREATE TRIGGER trg_after_ruta_update
AFTER UPDATE ON Ruta
FOR EACH ROW
BEGIN
    -- Solo actuar si el estatus realmente cambió
    IF OLD.estatus <> NEW.estatus THEN
        INSERT INTO LogRutas (idRuta, fechaCambio, usuario, estatusAnterior, estatusNuevo)
        VALUES (OLD.idRuta, NOW(), USER(), OLD.estatus, NEW.estatus);
    END IF;
END$$
DELIMITER ;

-- Este trigger se activa automáticamente. Ejemplo de una acción que lo dispara:
UPDATE Ruta SET estatus = 'Completada' WHERE idRuta = 1;
```

*Ejemplo 2: Impedir Asignación de Rutas a Conductores con Licencia Vencida*

Este trigger de tipo BEFORE actúa como una regla de negocio crítica, previniendo una operación inválida antes de que ocurra.

```
DELIMITER $$
CREATE TRIGGER trg_before_ruta_insert
BEFORE INSERT ON Ruta
FOR EACH ROW
BEGIN
    DECLARE estado_licencia VARCHAR(20);
```

```
    -- Obtenemos el estatus de la licencia del conductor a asignar
    SET estado_licencia = fn_estatus_licencia(NEW.idConductor);

    -- Si la licencia está vencida, cancelamos la inserción y lanzamos un error
    IF estado_licencia = 'Vencida' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: No se puede asignar la ruta. La licencia del conductor está vencida.';
    END IF;
END$$
DELIMITER ;

-- Ejemplo de una acción que sería bloqueada por el trigger:
-- INSERT INTO Ruta (idVehiculo, idConductor, ...) VALUES (1, [ID_de_conductor_con_licencia_vencida], ...);
-- La operación anterior fallaría y devolvería el mensaje de error definido.
```