Universidad Autónoma del Estado de Hidalgo

REPORTE DE PRÁCTICA NO. 1.5

PRACTICA 0

ALUMNO:
Morales Vázquez Rubén

1. Introducción

En este reporte de practica lo que vamos a realizar va a ser la practica cero, que consiste en buscar en físico o en digital sobre los temas de procedimientos almacenados, funciones, estructuras de control condicionales y repetitivas, disparadores, ademas de implementar ejercicios en nuestra base de datos ya antes hecha.

2. Herramientas empleadas

Las herramientas que fueron necesarias para la realización de esta practica fueron las siguientes:

1. Biblioteca digital. Plataforma web donde se encuentran libros y artículos donde podemos encontrar información.
2. MySQL Workbench. Es la aplicación donde logramos establecer la base de datos en lenguaje SQL.
3. Online LaTeX Editor Overleaf. Es un sitio web en el cual podemos generar nuestros documentos.

3. Marco teórico

Procedimientos almacenados

Un procedimiento almacenado (Stored Procedure) es un conjunto de instrucciones SQL que se almacena en la base de datos y se puede ejecutar cuando sea necesario. Se utiliza para encapsular lógica de negocio, mejorar el rendimiento y aumentar la seguridad en la gestión de datos.

Sintaxis de un Procedimiento Almacenado

```
DELIMITER //

CREATE PROCEDURE Nombre_Procedimiento()
BEGIN
    -- Instrucciones SQL
END //

DELIMITER ;
```

Funciones (Function)

Una función (FUNCTION) en SQL es un bloque de código almacenado en la base de datos que realiza un cálculo y devuelve un único valor. Se usa principalmente para encapsular lógica reutilizable, como cálculos matemáticos, manipulación de cadenas o transformación de datos, y puede ser invocada dentro de consultas SQL.
A diferencia de los procedimientos almacenados, las funciones siempre devuelven un valor y no pueden modificar datos en la base de datos (INSERT, UPDATE, DELETE).

Sintaxis de un Procedimiento Almacenado

```
DELIMITER //

CREATE FUNCTION Nombre_Funcion(parametros)
RETURNS tipo_dato
DETERMINISTIC
BEGIN
    -- Cuerpo de la funcion
    RETURN valor;
END //

DELIMITER ;
```

Estructuras de control condicionales y repetitivas

Las estructuras de control condicionales y repetitivas en SQL permiten controlar el flujo de ejecución dentro de procedimientos almacenados y funciones.

- Estructuras condicionales: Permiten ejecutar instrucciones según una condición específica. Ejemplos: IF... ELSE y CASE.
- Estructuras repetitivas: Permiten repetir un bloque de código mientras se cumpla una condición. Ejemplos: LOOP, WHILE y REPEAT... UNTIL.

Estas estructuras se utilizan para implementar lógica de negocio dentro de la base de datos, optimizando la ejecución de procesos.

3.3.1 Condicionales

Sintaxis de IF ELSE

```
IF condicion THEN
-- Codigo si la condicion es verdadera
ELSE
    -- Codigo si la condicion es falsa
END IF;
```

Sintaxis de CASE

```
SELECT
    columna1,
    columna2,
    CASE
        WHEN condicion1 THEN resultado1
        WHEN condicion2 THEN resultado2
        ELSE resultado_por_defecto
    END AS Nombre_Columna
FROM Tabla;
```

3.3.2 Repetitivas

Sintaxis de LOOP

```
DELIMITER //

CREATE PROCEDURE Nombre_Procedimiento()
BEGIN
    DECLARE contador INT DEFAULT 1;

    Nombre_Loop: LOOP
        -- Codigo a ejecutar en cada iteracion
        IF condicion_de_salida THEN
            LEAVE Nombre_Loop; -- Sale del bucle
        END IF;

        -- Incremento o cambios en variables
    END LOOP;

END //

DELIMITER ;
```

Sintaxis de WHILE

```
DELIMITER //

CREATE PROCEDURE Nombre_Procedimiento()
BEGIN
    DECLARE contador INT DEFAULT 1;

    WHILE condicion DO
        -- Codigo a ejecutar en cada iteracion
        SET contador = contador + 1; -- Incremento o cambios en variables
    END WHILE;

END //

DELIMITER ;
```

Sintaxis de REPEAT

```
DELIMITER //

CREATE PROCEDURE Nombre_Procedimiento()
BEGIN
    DECLARE contador INT DEFAULT 1;

    REPEAT
        -- Codigo a ejecutar en cada iteracion
        SET contador = contador + 1; -- Incremento o cambios en variables
    UNTIL condicion_de_salida END REPEAT;

END //

DELIMITER ;
```

Disparadores (Triggers)

Los disparadores (triggers) en SQL son bloques de código que se ejecutan automáticamente en respuesta a eventos específicos en una tabla, como INSERT, UPDATE o DELETE. Se utilizan para mantener la integridad de los datos, realizar auditorías o automatizar procesos dentro de la base de datos.

Sintaxis de TRIGGERS

```
CREATE TRIGGER Nombre_Trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON Nombre_Tabla
FOR EACH ROW
BEGIN
    -- Codigo a ejecutar cuando se activa el disparador
END;
```

Ejemplos implementados en la Base de Datos

3.5.1 Procedimientos almacenados

Inserta un nuevo auto en la tabla auto

```
DELIMITER //
CREATE PROCEDURE AgregarAuto(
    IN p_idauto INT,
    IN p_modelo VARCHAR(80),
    IN p_marca VARCHAR(80),
    IN p_anio INT,
    IN p_placa VARCHAR(20)
)
BEGIN
    INSERT INTO auto (idauto, modelo, marca, anio, placa)
    VALUES (p_idauto, p_modelo, p_marca, p_anio, p_placa);
END //
DELIMITER ;
```

Forma de usarlo

```
CALL AgregarAuto(6, 'Civic', 'Honda', 2022, 'ABC-1234');
```

Registrar un nuevo mantenimiento

```
DELIMITER //
CREATE PROCEDURE RegistrarMantenimiento(
    IN p_idmantenimiento INT,
    IN p_idAuto INT,
    IN p_fecha DATE,
    IN p_descripcion TEXT,
    IN p_costo DECIMAL(10,2)
)
BEGIN
    INSERT INTO mantenimiento (idmantenimiento, idAuto, fecha, descripcion,
        costo)
    VALUES (p_idmantenimiento, p_idAuto, p_fecha, p_descripcion, p_costo);
END //
DELIMITER ;
```

Forma de usarlo

```
CALL RegistrarMantenimiento(6, 1, '2025-01-10', 'Cambio de aceite y frenos',
    1200.50);
```

3.5.2 Funciones (Function)

Devuelve el año de un auto dado su idAuto.

```
DELIMITER //

CREATE FUNCTION ObtenerAnioAuto(p_idAuto INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE v_anio INT;

    SELECT anio INTO v_anio
    FROM auto
    WHERE idAuto = p_idAuto;

    RETURN v_anio;
END //

DELIMITER ;
```

Forma de usarlo

```
SELECT ObtenerAnioAuto(1);
```

Devuelve el total de autos registrados.

```
DELIMITER //
CREATE FUNCTION ContarAutos()
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE total INT;

    SELECT COUNT(*) INTO total FROM auto;

    RETURN total;
END //
DELIMITER ;
```

Forma de usarlo

```
SELECT ContarAutos();
```

3.5.3 Estructuras de control condicionales y repetitivas

Verifica si existe o no un auto en la tabla

```
DELIMITER //
CREATE PROCEDURE VerificarAuto(IN p_idAuto INT)
BEGIN
    DECLARE existe INT;

    -- Verificar si el auto existe
    SELECT COUNT(*) INTO existe FROM auto WHERE idAuto = p_idAuto;

    IF existe > 0 THEN
        SELECT 'El auto existe en la base de datos' AS Resultado;
    ELSE
        SELECT 'El auto NO existe en la base de datos' AS Resultado;
    END IF;
END //
DELIMITER ;
```

Forma de usarlo

```
CALL VerificarAuto(1);
CALL VerificarAuto(100);
```

Muestra los IDs de los primeros 5 autos

```
DELIMITER //
CREATE PROCEDURE ContarAutos()
BEGIN
    DECLARE contador INT DEFAULT 1;

    loop_label: LOOP
        IF contador > 5 THEN
            LEAVE loop_label; -- Salir del loop cuando llega a 5
        END IF;

        SELECT idAuto AS ID FROM auto WHERE idAuto = contador;
        SET contador = contador + 1;
    END LOOP;
END //
DELIMITER ;
```

Forma de usarlo

```
CALL ContarAutos();
```

3.5.4 Disparadores (Triggers)

Si el número es negativo, lo cambia a 0.0

```
DELIMITER //

CREATE TRIGGER before_insert_mantenimiento
BEFORE INSERT ON mantenimiento
FOR EACH ROW
BEGIN
    IF NEW.costo < 0 THEN
        SET NEW.costo = 0;
    END IF;
END //

DELIMITER ;
```

Forma de usarlo

```
INSERT INTO mantenimiento (idMantenimiento, fecha, descripcion, costo, idAuto)
VALUES (1, '2025-02-01', 'Cambio de aceite', -100, 1);

SELECT * FROM mantenimiento;
```

Verifica que la fecha no sea mayor a la actual

```
DELIMITER //
CREATE TRIGGER before_insert_registro_conduccion
BEFORE INSERT ON registro_conduccion
FOR EACH ROW
BEGIN
    IF NEW.fecha > CURDATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No se pueden registrar conducciones en el futuro';
    END IF;
END //

DELIMITER ;
```

Forma de usarlo

```
-- Esto dara error porque la fecha es en el futuro
INSERT INTO registro_conduccion (idRegistro, fecha, idConductor, idAuto)
VALUES (2, '2030-01-01', 1, 1);
```

4. Conclusiones

En esta practica 0 lo que aprendimos fue a utilizar la biblioteca digital y buscar información sobre los prodedimientos almacenados, funciones, estructuras de control y repetitivas, y por ultimo los disparadores, despues de buscar la información y como estan estructurados con su sintaxis logramos implementarlos en nuestra base de datos de Flotilla de Autos en donde aplicamos ejemplos donde utilizamos todas las estructuras investigadas, fue complicado en algunas cosas por el uso de la sintaxis ya que tiene diferentes sintaxis para cada lenguaje, pero en SQL lo logramos implementar de manera correcta.

References

[1] Giménez, J. A. (**2019**). *Buenas prácticas en el diseño de bases de datos.* ARANDU UTIC, *6*(1), 193–210.

[2] Mosquera Palacios, D. J., & Wanumen Silva, L. F. (**2018**). *Arquitectura para la generación de consultas SQL usando lógica de conjuntos.* Visión Electrónica, *2*, 307–318.

[3] Date, C. J. (**2004**). An Introduction to Database Systems *8th ed Addison-Wesley*

[4] Elmasri, R. Navathe, S. B. (**2016**). Fundamentals of Database Systems *11th ed Pearson*