REPORTE DE PRÁCTICA NO. 0

NOMBRE DE LA PRÁCTICA: Practica 0 : Gestion de Flotilla de Autos

ALUMNO: Saúl Jiménez Mercado

Dr. Eduardo Cornejo-Velázquez

1. Introducción

En esta práctica, nos apoyamos en la biblioteca digital y diversas fuentes de información para investigar y comprender el uso de procedimientos almacenados, funciones, estructuras de control condicionales y repetitivas, así como disparadores en bases de datos relacionales. A través de este estudio, buscamos analizar cómo estas herramientas permiten automatizar tareas, mejorar la eficiencia en el manejo de la información y garantizar la integridad de los datos dentro del sistema de gestión de flotillas. Además, esta exploración nos ayuda a asimilar la lógica de programación aplicada a bases de datos, estableciendo paralelismos con otros lenguajes de programación estructurada y fortaleciendo nuestro conocimiento en el diseño de soluciones optimizadas para entornos transaccionales.

2. Marco teórico

Procedimientos almacenados (Procedure)

Los procedimientos almacenados (stored procedures) son programas autocontrolados, reutilizables, escritos en lenguaje del DBMS, se encuentran almacenados como parte de la base de datos y sus metadatos. Un procedimiento almacenado contiene código SQL que puede, entre otras cosas, insertar, actualizar o eliminar registros. También puede alterar la estructura de la base de datos, por ejemplo, puede añadir una columna de tabla o borrar una tabla.
Es un objeto que se crea con la sentencia CREATE PROCEDURE y se invoca con la sentencia CALL. Un procedimiento puede tener cero o muchos parámetros de entrada y cero o muchos parámetros de salida.

Funciones (Function)

Es un objeto que se crea con la sentencia CREATE FUNCTION y se invoca con la sentencia SELECT o dentro de una expresión. Las funciones pueden aceptar parámetros y devuelven siempre un valor. Este valor devuelto puede ser un valor nulo y en este caso se comportaría como un procedimiento.

Estructuras de control condicionales y repetitivas

Las estructuras de control permiten, como su nombre lo indica, controlar el flujo de las instrucciones dentro de un procedimiento o una funci´on. Algunas de las estructuras pueden llevar una etiqueta (BEGIN, LOOP, REPEAT y WHILE).

Disparadores (Triggers)

Un trigger es un tipo especial de procedimiento almacenado, en términos generales las características de un procedimiento almacenado se cumplen también con los triggers, son rutinas autónomas asociadas a una tabla, (algunos DBMS aceptan asociar triggers a vistas), con la especial diferencia de que se ponen en funcionamiento automáticamente como respuesta a ciertas modificaciones de los datos, como inserciones, actualizaciones o eliminaciones; es decir, que el trigger se autoejecuta cuando los valores de la base de datos cumplen con cierta condición explícita al momento de la creación del trigger
Un trigger es un objeto de la base de datos que está asociado con una tabla y que se activa cuando ocurre un evento sobre la tabla. Los eventos que pueden ocurrir sobre la tabla son:

1. INSERT: El trigger se activa cuando se inserta una nueva fila sobre la tabla asociada.
2. UPDATE: El trigger se activa cuando se actualiza una fila sobre la tabla asociada.
3. DELETE: El trigger se activa cuando se elimina una fila sobre la tabla asociada.

3. Herramientas empleadas

Describir qué herramientas se han utilizado...

1. MySQL Server: MySQL Workbench es una herramienta visual unificada para arquitectos de bases de datos, desarrolladores y administradores de bases de datos. MySQL Workbench proporciona modelado de datos, desarrollo de SQL y herramientas de administración integrales para configuración de servidores, administración de usuarios, copias de seguridad y mucho más. Permite diseñar, visualizar y mantener esquemas de bases de datos de manera eficiente.

4. Desarrollo

Procedimientos almacenados (Procedure)

Listing 1: Registrar Mantenimiento

```
DELIMITER //
CREATE PROCEDURE RegistrarMantenimiento(
    IN p_id_Vehiculo INT,
    IN p_fecha DATE,
    IN p_tipo VARCHAR(80),
    IN p_costo INT
)
BEGIN
    -- Verificar que el costo sea mayor a 0
    IF p_costo <= 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'El costo del mantenimiento debe ser mayor a 0';
    END IF;

    -- Insertar el mantenimiento
    INSERT INTO Mantenimiento (id_Mantenimiento, fecha, tipo,
    costo, id_Vehiculo)
    VALUES (NULL, p_fecha, p_tipo, p_costo, p_id_Vehiculo);

    -- Mostrar los mantenimientos registrados para ese veh culo
    SELECT * FROM Mantenimiento WHERE id_Vehiculo = p_id_Vehiculo;
END //
DELIMITER ;
```

Listing 2: Actualizar Estado Vehiculos

```
DELIMITER //
CREATE PROCEDURE ActualizarEstadoVehiculos()
BEGIN
    UPDATE Auto
    SET estado = 'Requiere Mantenimiento'
    WHERE id_Vehiculo IN (SELECT id_Vehiculo FROM Mantenimiento WHERE fecha <=
END //
DELIMITER ;
```

Funciones (Function)

Listing 3: Gasto Total Combustible

```
DELIMITER //
CREATE FUNCTION GastoTotalCombustible(p_id_Vehiculo INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(10,2);

    SELECT SUM(costo) INTO total
    FROM Gasolina
    WHERE id_Vehiculo = p_id_Vehiculo;

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

SELECT GastoTotalCombustible(2);
```

Listing 4: Kilometraje Promedio Vehiculo

```
DELIMITER //
CREATE FUNCTION KilometrajePromedioVehiculo(p_id_Vehiculo INT) RETURNS DECIMAL
BEGIN
    DECLARE promedio DECIMAL(10,2);

    -- Calcular el kilometraje promedio
    SELECT AVG(distancia) INTO promedio
    FROM Ruta
    WHERE id_Vehiculo = p_id_Vehiculo;

    -- Si no hay rutas asociadas, devolver 0
    IF promedio IS NULL THEN
        SET promedio = 0;
    END IF;

    RETURN promedio;
END //
DELIMITER ;
```

Estructuras de control condicionales y repetitivas)

Listing 5: Actualizar Estatus Vehiculos

```
DELIMITER //
CREATE PROCEDURE ActualizarEstatusVehiculos()
BEGIN
    DECLARE vehiculo_id INT;
    DECLARE fecha_ultimo_mantenimiento DATE;

    -- Cursor para obtener los veh culos y su  ltimo  mantenimiento
    DECLARE cur CURSOR FOR
        SELECT id_Vehiculo, MAX(fecha) FROM Mantenimiento
        GROUP BY id_Vehiculo;

    -- Manejo de fin de cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;

    revisar_loop: LOOP
        FETCH cur INTO vehiculo_id, fecha_ultimo_mantenimiento;
        IF done THEN
            LEAVE revisar_loop;
        END IF;

        -- Condicional para verificar si el vehiculo requiere mantenimiento
        IF fecha_ultimo_mantenimiento <= DATE_SUB(NOW(), INTERVAL 6 MONTH)
        THEN
            UPDATE Auto SET estado = 'Requiere Mantenimiento'
            WHERE id_Vehiculo = vehiculo_id;
        ELSE
            UPDATE Auto SET estado = 'En Buen Estado'
            WHERE id_Vehiculo = vehiculo_id;
        END IF;
    END LOOP;

    CLOSE cur;
END //
DELIMITER ;
```

Listing 6: Asignar Turnos

```
DELIMITER //
CREATE PROCEDURE AsignarTurnos()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE conductor_id INT;

    -- Cursor para obtener los conductores sin turno asignado
    DECLARE cur CURSOR FOR
        SELECT id_Conductor FROM Conductor
        WHERE id_Conductor NOT IN (SELECT id_Conductor
        FROM RegistroTurno WHERE fecha = CURDATE());
```

```
    -- Manejo de fin de cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;

    asignar_loop: LOOP
        FETCH cur INTO conductor_id;
        IF done THEN
            LEAVE asignar_loop;
        END IF;

        -- Asignar un turno a cada conductor
        INSERT INTO RegistroTurno (id_Conductor, fecha, horario)
        VALUES (conductor_id, CURDATE(), '08:00 - 16:00');
    END LOOP;

    CLOSE cur;
END //
DELIMITER ;
```

Disparadores (Trigger)

Listing 7: Registrar Ultima Verificacion

```
DELIMITER //
CREATE TRIGGER RegistrarUltimaVerificacion
AFTER UPDATE ON Documentos
FOR EACH ROW
BEGIN
    UPDATE Auto
    SET ultima_verificacion = NOW()
    WHERE id_Vehiculo = NEW.id_Vehiculo;
END //
DELIMITER ;
```

Listing 8: Generar Mantenimiento Preventivo

```
DELIMITER //
CREATE TRIGGER GenerarMantenimientoPreventivo
AFTER UPDATE ON Auto
FOR EACH ROW
BEGIN
    IF NEW.ano <= DATE_SUB(NOW(), INTERVAL 5 YEAR) THEN
        INSERT INTO Mantenimiento (id_Mantenimiento, fecha, tipo,
        costo, id_Vehiculo)
        VALUES (NULL, NOW(),
        'Mantenimiento Preventivo', 2500, NEW.id_Vehiculo);
    END IF;
END //
DELIMITER ;
```

5. Conclusiones

Con esta práctica aprendí que existen múltiples enfoques para gestionar los datos dentro de una base de datos, lo que me permite tener un control más preciso sobre la información. A través de herramientas como los procedimientos almacenados y las funciones, puedo automatizar procesos y realizar cálculos complejos de manera eficiente. Además, el uso de estructuras condicionales y repetitivas me permite aplicar lógica dentro de las consultas, lo que optimiza la manera en que actualizo o filtro los datos. De igual forma, los disparadores son útiles para asegurar que ciertas acciones ocurran automáticamente bajo condiciones específicas. En general, estas técnicas no solo facilitan la manipulación de los datos, sino que también mejoran la integridad y la seguridad de la base de datos, permitiendo realizar operaciones más complejas y eficientes.

Referencias Bibliográficas

References

[1] UNIDAD VII Introducción al SQL procedural. (n.d.). https://tigger.celaya.tecnm.mx/conacad/cargas/GAFR590328RX9/55/Unidad$_V II_s ql_p rocedural.pdf$

[2] Hernández, J. J. S. (s. f.). Unidad 12. Triggers, procedimientos y funciones en MySQL. https://josejuansanchez.org/bd/unidad-12-teoria/index.htmlfunciones