Universidad Autónoma del Estado de Hidalgo

REPORTE DE PRÁCTICA NO 1.5

PRÁCTICA 0

ALUMNO: Esperilla Mendoza Luis Erick
Dr. Eduardo Cornejo-Velázquez

1. Introducción

En el ámbito de la gestión de bases de datos, es fundamental implementar mecanismos que optimicen la ejecución de procesos, mejoren la seguridad y garanticen la integridad de los datos. En esta práctica, se explorarán conceptos clave como los procedimientos almacenados, funciones, estructuras de control condicionales y repetitivas, y disparadores.

Estas herramientas permiten automatizar tareas, mejorar el rendimiento de las consultas y mantener la consistencia de la información en una base de datos.

Al finalizar la actividad, se generarán dos entregables principales: el archivo de respaldo de la base de datos y el reporte de la práctica. Estos documentos se subirán al repositorio de GitHub, y posteriormente se registrará la URL correspondiente en la plataforma Garza como evidencia de la actividad.

2. Marco teórico

Procedimientos almacenados (Procedure)

Los procedimientos almacenados son conjuntos de instrucciones SQL que se almacenan en el servidor de base de datos y se ejecutan bajo demanda. Se utilizan para encapsular lógica compleja, reducir la repetición de código y mejorar la eficiencia en la ejecución de consultas.

Beneficios:
- Reducción de tráfico entre la aplicación y la base de datos.
- Mayor seguridad y control de acceso.
- Mejor rendimiento al optimizar consultas precompiladas.
- Facilitan la modularidad y el mantenimiento del código.

Funciones (Function)

Las funciones en SQL permiten encapsular lógica que devuelve un valor específico. A diferencia de los procedimientos almacenados, una función siempre devuelve un resultado y no puede modificar los datos directamente en la base de datos.

Características:
- Devuelven un solo valor.
- No pueden ejecutar operaciones INSERT, UPDATE o DELETE.
- Son reutilizables dentro de consultas SQL.

- Mejoran la legibilidad y organización del código.

Estructuras de control condicionales y repetitivas

Las estructuras de control permiten ejecutar instrucciones de manera condicional o repetitiva, optimizando la ejecución de consultas y procesos en la base de datos.

Tipos:

Condicionales:
- IF...THEN...ELSE: Evalúa condiciones y ejecuta instrucciones según el resultado.
- CASE: Similar a IF, pero útil para evaluar múltiples condiciones.

Repetitivas:

- LOOP: Ejecuta un bloque de código repetidamente hasta que se cumpla una condición.
- WHILE: Repite instrucciones mientras una condición sea verdadera.
- REPEAT...UNTIL: Ejecuta un bloque hasta que se cumpla una condición de finalización.

Disparadores (Triggers)

Los disparadores son procedimientos especiales que se ejecutan automáticamente en respuesta a eventos como INSERT, UPDATE o DELETE en una tabla. Se utilizan para garantizar la integridad de los datos y la automatización de procesos.

Beneficios:

- Aplican reglas de negocio automáticamente.
- Mantienen la consistencia de los datos.
- Reducen la intervención manual en la gestión de registros.
- Permiten auditar cambios en la base de datos.

Con estos conceptos y ejemplos aplicados a la base de datos de gestión de flotillas, se puede optimizar el manejo de información, mejorar la eficiencia y garantizar la integridad de los datos.

3. Herramientas empleadas

- **DataGrip**
  - **Tipo**: IDE para bases de datos desarrollado por JetBrains.
  - **Uso**: Se utiliza para gestionar y administrar bases de datos, escribir y ejecutar consultas SQL, diseñar esquemas de bases de datos, y conectarse a múltiples servidores de bases de datos, todo en una sola interfaz.

4. Desarrollo

Procedimientos almacenados (Procedure)

Ejemplo 1: Registrar un nuevo conductor
Este procedimiento inserta un nuevo conductor en la tabla Conductor.

```
DELIMITER //

    CREATE PROCEDURE InsertarConductor(
        IN p_idConductor INT,
        IN p_nombre VARCHAR(80),
        IN p_apellido VARCHAR(80)
    )
BEGIN
    INSERT INTO Conductor (idConductor, nombre, apellido)
    VALUES (p_idConductor, p_nombre, p_apellido);
END //

DELIMITER ;
```

Llamada al procedimiento:

```
CALL InsertarConductor(6, 'Damian', 'Ortiz');
```

Ejemplo 2: Obtener registros laborales de un conductor en una fecha específica

```
DELIMITER //

    CREATE PROCEDURE ObtenerRegistroLaboral(
    IN p_idConductor INT,
    IN p_fecha DATE
    )
BEGIN
    SELECT * FROM RegistroLaboral
    WHERE idConductor = p_idConductor AND fechaTrabajo = p_fecha;
END //

DELIMITER ;
```

Llamada al procedimiento:

```
CALL ObtenerRegistroLaboral(1, '2024-02-25');
```

Funciones (Function)

Ejemplo 1: Calcular el costo total de combustible de un vehículo
Esta función calcula el total gastado en combustible por un vehículo específico.

```
DELIMITER //

    CREATE FUNCTION CalcularCostoTotalCombustible(p_idVehiculo INT)
    RETURNS FLOAT DETERMINISTIC
```

```
BEGIN
    DECLARE total FLOAT;
    SELECT SUM(totalCosto) INTO total
    FROM detalle_combustible
    WHERE idVehiculo = p_idVehiculo;
    RETURN COALESCE(total, 0);
END //

DELIMITER ;
```

Uso de la función:

```
SELECT CalcularCostoTotalCombustible(1);
```

Ejemplo 2: Obtener la cantidad de mantenimientos de un vehículo

```
DELIMITER //

CREATE FUNCTION ContarMantenimientos(p_idVehiculo INT)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE cantidad INT;
    SELECT COUNT(*) INTO cantidad
    FROM Mantenimiento
    WHERE idVehiculo = p_idVehiculo;
    RETURN cantidad;
END //

DELIMITER ;
```

Uso de la función:

```
SELECT ContarMantenimientos(1);
```

Estructuras de Control Condicionales y Repetitivas

Ejemplo 1: Verificar si un vehículo tiene conductor asignado (IF...ELSE)

```
DELIMITER //

    CREATE PROCEDURE VerificarConductorAsignado(IN p_idVehiculo INT)
    BEGIN
        DECLARE conductorAsignado INT;

        SELECT idConductor INTO conductorAsignado
        FROM Vehiculo
        WHERE idVehiculo = p_idVehiculo;

    IF conductorAsignado IS NOT NULL THEN
        SELECT 'El vehículo tiene un conductor asignado' AS Mensaje;
    ELSE
        SELECT 'El vehículo NO tiene un conductor asignado' AS Mensaje;
    END IF;
END //
```

```
DELIMITER ;
```

Llamada al procedimiento:

```
CALL VerificarConductorAsignado(1);
```

Ejemplo 2: Mostrar los conductores con más de 3 registros laborales (WHILE loop)

```
DELIMITER //

CREATE PROCEDURE ConductoresFrecuentes()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE conductor_id INT;
    DECLARE cur CURSOR FOR
        SELECT idConductor FROM RegistroLaboral
        GROUP BY idConductor
        HAVING COUNT(*) > 3;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;

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

        SELECT * FROM Conductor WHERE idConductor = conductor_id;
    END LOOP;

    CLOSE cur;
END //

DELIMITER ;
```

Llamada al procedimiento:

```
CALL ConductoresFrecuentes();
```

Disparadores (Triggers)

Ejemplo 1: Evitar la inserción de registros laborales con horas de salida anteriores a la entrada

```
DELIMITER //

CREATE TRIGGER ValidarHorasTrabajo
BEFORE INSERT ON RegistroLaboral
FOR EACH ROW
BEGIN
```

```
    IF NEW.horaSalida <= NEW.horaEntrada THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La hora de salida debe ser mayor que la hora de entrada';
    END IF;
END //

DELIMITER ;
```

Intento de inserción incorrecta:

```
    INSERT INTO RegistroLaboral (idConductor, fechaTrabajo, horaEntrada, horaSalida,
    ↪  tipoTrabajo, idVehiculo, observaciones)
        VALUES (1, '2024-02-25', '15:00:00', '14:00:00', 'Transporte', 2, 'Error en la
        ↪  inserción');
```

Ejemplo 2: Registrar automáticamente el costo total en detalleCombustible

```
DELIMITER //

    CREATE TRIGGER CalcularTotalCombustible
    BEFORE INSERT ON detalle_combustible
    FOR EACH ROW
BEGIN
SET NEW.totalCosto = NEW.cantidadLitros * NEW.precioPorLitro;
END //

DELIMITER ;
```

Inserción de datos:

```
    INSERT INTO detalle_combustible (idDetalleCombustible, fechaCarga, cantidadLitros,
    ↪  precioPorLitro, idVehiculo)
        VALUES (1, '2024-02-25', 50, 22.5, 1);
```

5. Conclusiones

En esta práctica reforcé mis conocimientos en SQL mediante la implementación de procedimientos almacenados, funciones, estructuras de control y triggers en la base de datos de gestión de flotillas.

Aprendí a automatizar procesos, optimizar consultas y garantizar la integridad de los datos. Me sorprendió cómo los triggers pueden prevenir errores automáticamente.

En general, mejoré mis habilidades en administración de bases de datos y optimización de procesos.

Referencias Bibliográficas

References

[1] Coronel, C., & Morris, S. (2020). *Bases de datos: diseño, implementación y administración* (13ª ed.). Cengage Learning.

[2] Elmasri, R., & Navathe, S. (2017). *Sistemas de bases de datos* (7ª ed.). Pearson.

[3] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Fundamentos de bases de datos* (7ª ed.). McGraw-Hill.

[4] González, G. (2018). Procedimientos almacenados y funciones en SQL. *Revista de Ingeniería y Tecnología*, *25*(2), 45-60.

[5] Ramírez, J. (2021). *SQL avanzado: Triggers, procedimientos y optimización de consultas*. Alfaomega.

[6] Martínez, L. (2019). Disparadores en bases de datos relacionales: fundamentos y aplicaciones. *Tecnología y Desarrollo*, *12*(1), 78-92.

[7] García, P. (2016). *Estructuras de control en SQL: Condiciones y bucles*. Editorial Científica.