25/09/2025

Practica 0
Biblioteca

Bases de Datos Distribuidas

Cesar Rodriguez Garcia

1. Procedimientos Almacenados (*Stored Procedures*)

Concepto	Definición
Procedimiento Almacenado	Es un **conjunto de sentencias SQL** (incluyendo lógica de control como IF o WHILE) que se guarda y almacena directamente en el motor de la base de datos. Se ejecuta bajo demanda con el comando EXECUTE o CALL.
Características	🔹 **Reutilización:** Se llama muchas veces desde distintas aplicaciones. 🔹 **Rendimiento:** Se compilan una sola vez (se pre-analizan) y se ejecutan más rápido. 🔹 **Seguridad:** Permite a los usuarios interactuar con los datos sin tener acceso directo a las tablas. 🔹 **Modularidad:** Facilita la programación estructurada y el mantenimiento.

Ejemplo de Construcción (Gestión de Flotillas)

Objetivo: Registrar un nuevo mantenimiento y actualizar el kilometraje del vehículo en una sola operación atómica.

```
-- Procedimiento Almacenado: RegistrarMantenimientoYActualizarKM

CREATE PROCEDURE RegistrarMantenimientoYActualizarKM

    @VehiculoID INT,

    @TipoMantenimiento VARCHAR(100),

    @Costo DECIMAL(10, 2),

    @KilometrajeNuevo INT

AS

BEGIN

    -- 1. Insertar el registro de mantenimiento

    INSERT INTO Mantenimientos (VehiculoID, Fecha, Tipo, Costo,
KilometrajeMantenimiento)
```

```
    VALUES (@VehiculoID, GETDATE(), @TipoMantenimiento, @Costo,
@KilometrajeNuevo);


    -- 2. Actualizar el kilometraje actual del vehículo
    UPDATE Vehiculos
    SET Kilometraje = @KilometrajeNuevo
    WHERE VehiculoID = @VehiculoID;


    -- Opcional: Devolver un mensaje de éxito
    SELECT 'Mantenimiento registrado y kilometraje actualizado correctamente.' AS
Resultado;
END;
```

2. Funciones (*Functions*)

Concepto	**Definición**
Función	Es un **conjunto de sentencias SQL** (y lógica de control) que realiza un cálculo y **devuelve un único valor o una tabla**. Las funciones son llamadas dentro de una expresión SQL (como `SELECT`, `WHERE`, o `HAVING`).
Características	🔹 **Reutilización:** Se pueden usar en cualquier lugar donde se acepte una expresión. 🔹 **Devuelve Valor:** Obligatoriamente retorna un valor (escalar) o una tabla (tabular). 🔹 **Sin Alteración de Datos:** Por lo general, no pueden realizar operaciones de modificación de datos (`INSERT`, `UPDATE`, `DELETE`) en las tablas.

```
-- Función: CalcularAntiguedadVehiculo
CREATE FUNCTION CalcularAntiguedadVehiculo (@VehiculoID INT)
```

```
RETURNS INT
AS
BEGIN
    DECLARE @AnioFabricacion INT;
    DECLARE @Antiguedad INT;

    -- Obtener el año de fabricación del vehículo
    SELECT @AnioFabricacion = AnioFabricacion
    FROM Vehiculos
    WHERE VehiculoID = @VehiculoID;

    -- Calcular la antigüedad en años
    SET @Antiguedad = YEAR(GETDATE()) - @AnioFabricacion;

    RETURN @Antiguedad;
END;

-- Ejemplo de uso:
-- SELECT dbo.CalcularAntiguedadVehiculo(101) AS AntiguedadEnAnios;
```

3. Estructuras de Control Condicionales y Repetitivas

Concepto	**Definición**
Condicionales	Sentencias que ejecutan un bloque de código **solo si se cumple una condición** especificada. Las más comunes son `IF-THEN-ELSE` y `CASE`.

Repetitivas	Sentencias que ejecutan un bloque de código **repetidamente** mientras se cumpla una condición o hasta que se cumpla una condición de salida. La más común es WHILE.
Uso	Estas estructuras se utilizan principalmente **dentro** de Procedimientos Almacenados y Funciones para implementar lógica de negocio compleja.

Claro. A continuación, se presenta el marco teórico y dos ejemplos de construcción para cada uno de los conceptos solicitados, aplicados al contexto de una base de datos de **Gestión de Flotillas** (tablas sugeridas: Vehiculos, Mantenimientos, Conductores, Asignaciones).

Marco Teórico y Ejemplos de Programación en Bases de Datos

1. Procedimientos Almacenados (*Stored Procedures*) 💾

Concepto	**Definición**
Procedimiento Almacenado	Es un **conjunto de sentencias SQL** (incluyendo lógica de control como IF o WHILE) que se guarda y almacena directamente en el motor de la base de datos. Se ejecuta bajo demanda con el comando EXECUTE o CALL.
Características	🔹 **Reutilización:** Se llama muchas veces desde distintas aplicaciones. 🔹 **Rendimiento:** Se compilan una sola vez (se pre-analizan) y se ejecutan más rápido. 🔹 **Seguridad:** Permite a los usuarios interactuar con los datos sin tener acceso directo a las tablas. 🔹 **Modularidad:** Facilita la programación estructurada y el mantenimiento.

Ejemplo de Construcción (Gestión de Flotillas)

Objetivo: Registrar un nuevo mantenimiento y actualizar el kilometraje del vehículo en una sola operación atómica.

SQL

```
-- Procedimiento Almacenado: RegistrarMantenimientoYActualizarKM

CREATE PROCEDURE RegistrarMantenimientoYActualizarKM

    @VehiculoID INT,

    @TipoMantenimiento VARCHAR(100),

    @Costo DECIMAL(10, 2),

    @KilometrajeNuevo INT

AS

BEGIN

    -- 1. Insertar el registro de mantenimiento

    INSERT INTO Mantenimientos (VehiculoID, Fecha, Tipo, Costo,
KilometrajeMantenimiento)

    VALUES (@VehiculoID, GETDATE(), @TipoMantenimiento, @Costo,
@KilometrajeNuevo);


    -- 2. Actualizar el kilometraje actual del vehículo

    UPDATE Vehiculos

    SET Kilometraje = @KilometrajeNuevo

    WHERE VehiculoID = @VehiculoID;


    -- Opcional: Devolver un mensaje de éxito

    SELECT 'Mantenimiento registrado y kilometraje actualizado correctamente.' AS
Resultado;

END;
```

2. Funciones (*Functions*) 🔢

Concepto	**Definición**

Función	Es un **conjunto de sentencias SQL** (y lógica de control) que realiza un cálculo y **devuelve un único valor o una tabla**. Las funciones son llamadas dentro de una expresión SQL (como SELECT, WHERE, o HAVING).
Características	🔹 **Reutilización:** Se pueden usar en cualquier lugar donde se acepte una expresión. 🔹 **Devuelve Valor:** Obligatoriamente retorna un valor (escalar) o una tabla (tabular). 🔹 **Sin Alteración de Datos:** Por lo general, no pueden realizar operaciones de modificación de datos (INSERT, UPDATE, DELETE) en las tablas.

Ejemplo de Construcción (Gestión de Flotillas)

Objetivo: Calcular la antigüedad de un vehículo en años.

SQL

```
-- Función: CalcularAntiguedadVehiculo
CREATE FUNCTION CalcularAntiguedadVehiculo (@VehiculoID INT)
RETURNS INT
AS
BEGIN
    DECLARE @AnioFabricacion INT;
    DECLARE @Antiguedad INT;

    -- Obtener el año de fabricación del vehículo
    SELECT @AnioFabricacion = AnioFabricacion
    FROM Vehiculos
    WHERE VehiculoID = @VehiculoID;

    -- Calcular la antigüedad en años
    SET @Antiguedad = YEAR(GETDATE()) - @AnioFabricacion;

    RETURN @Antiguedad;
END;
```

-- Ejemplo de uso:

-- SELECT dbo.CalcularAntiguedadVehiculo(101) AS AntiguedadEnAnios;

---

3. Estructuras de Control Condicionales y Repetitivas 🚦🔄

Concepto	Definición
Condicionales	Sentencias que ejecutan un bloque de código **solo si se cumple una condición** especificada. Las más comunes son IF-THEN-ELSE y CASE.
Repetitivas	Sentencias que ejecutan un bloque de código **repetidamente** mientras se cumpla una condición o hasta que se cumpla una condición de salida. La más común es WHILE.
Uso	Estas estructuras se utilizan principalmente **dentro** de Procedimientos Almacenados y Funciones para implementar lógica de negocio compleja.

Ejemplo de Construcción (Gestión de Flotillas)

Objetivo: Determinar el estado de servicio de un vehículo basándose en su kilometraje actual (Condicional IF).

```
-- Este ejemplo se usaría DENTRO de un Procedimiento Almacenado o Función
CREATE PROCEDURE DeterminarEstadoServicio
    @VehiculoID INT
AS
BEGIN
    DECLARE @Kilometraje INT;
    DECLARE @EstadoServicio VARCHAR(50);
```

```
    -- Obtener el kilometraje actual del vehículo
    SELECT @Kilometraje = Kilometraje
    FROM Vehiculos
    WHERE VehiculoID = @VehiculoID;

    -- Estructura Condicional IF-ELSE-IF
    IF @Kilometraje > 200000
        SET @EstadoServicio = 'Requiere Revisión Mayor (Alto KM)';
    ELSE IF @Kilometraje >= 150000 AND @Kilometraje <= 200000
        SET @EstadoServicio = 'Servicio Programado';
    ELSE
        SET @EstadoServicio = 'Operativo Normal';

    SELECT @EstadoServicio AS EstadoActual;
END;
```

4. Disparadores (*Triggers*)

Concepto	**Definición**
Disparador	Es un tipo especial de procedimiento almacenado que se ejecuta **automáticamente** (se "dispara") en respuesta a un **evento de modificación de datos** (operaciones DML: INSERT, UPDATE, DELETE) en una tabla específica.
Características	🔹 **Ejecución Automática:** No requiere una llamada explícita. 🔹 **Eventos:** Se asocia a uno o más eventos (ej. AFTER INSERT, BEFORE UPDATE). 🔹 **Auditoría/Integridad:** Se usa

	comúnmente para mantener la integridad referencial compleja o para auditar cambios en los datos.

Ejemplo de Construcción (Gestión de Flotillas)

Objetivo: Auditar los cambios en las asignaciones de vehículos a conductores para mantener un historial.

```
-- Disparador: LogAuditoriaAsignacion

CREATE TRIGGER LogAuditoriaAsignacion

ON Asignaciones

AFTER UPDATE

AS

BEGIN

    -- Se dispara automáticamente DESPUÉS de una operación UPDATE en la tabla Asignaciones



    -- Verificar si se cambió la fecha de fin o el conductor

    IF UPDATE(ConductorID) OR UPDATE(FechaFin)

    BEGIN

        INSERT INTO LogAsignaciones (VehiculoID, ConductorAnteriorID, ConductorNuevoID, FechaCambio)

        SELECT

            d.VehiculoID,

            d.ConductorID AS ConductorAnterior,

            i.ConductorID AS ConductorNuevo,

            GETDATE()

        FROM

            DELETED d  -- La tabla 'DELETED' contiene los datos antes del UPDATE

        INNER JOIN

            INSERTED i ON d.AsignacionID = i.AsignacionID; -- La tabla 'INSERTED' contiene los datos después del UPDATE
```

```
    END
END;
```