Bases de Datos

Ayudantía 4: SQL Procedural

Profesor: Víctor Reyes

Ayudantes: Diego Banda, Felipe Gutiérrez

Contactos

Diego Banda

diego.banda@mail.udp.cl

darklouds

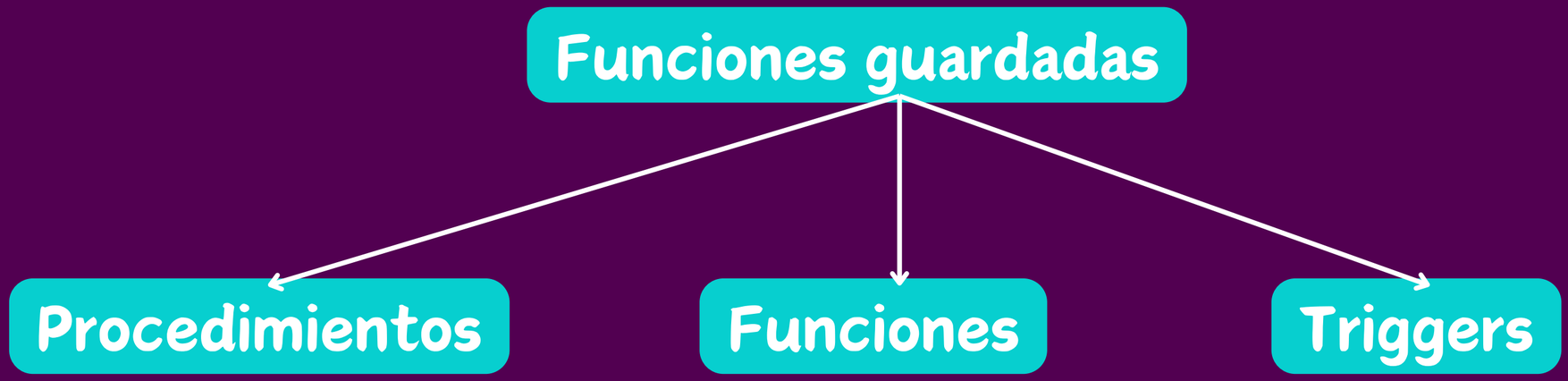
Felipe Gutiérrez

felipe.gutierrez_l@mail.udp.cl

felipx#0485

“Funciones guardadas”

Es como guardar bloques de código SQL para ser ejecutados posteriormente bajo distintas circunstancias.



Procedure

- Propósito: Es un procedimiento almacenado, conjunto de instrucciones SQL que se almacenan y pueden ejecutar varias veces. Se utilizan normalmente para llamados a varias tablas o cálculos.
- Uso: Se llama a través de CALL <nombre> y no necesariamente retorna un valor.

Características

- Para realizar operaciones complejas.
- Solo se llama, no se puede usar en consulta SQL.
- Puede tener parámetros de entrada (IN), de salida (OUT) y de entrada/salida (INOUT)

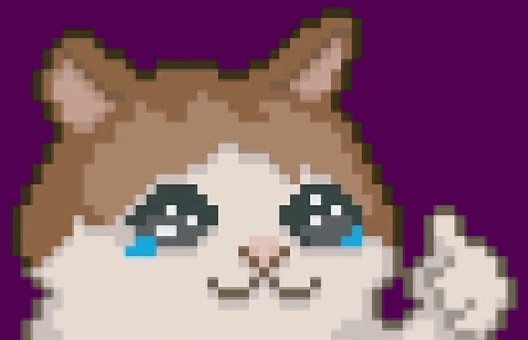
```
DELIMITER //
CREATE PROCEDURE actualizar_precio(producto_id INT, nuevo_precio INT)
BEGIN
    UPDATE productos SET precio = nuevo_precio WHERE id = producto_id;
END;
//
DELIMITER;
```

Actualiza precio de producto en id específico al nuevo precio entregado

Procedure sintaxis

- CREATE PROCEDURE <nombre>: Define y crea una nueva procedure.
- IN, OUT, INOUT: Define si una variable es de entrada (IN), salida (OUT) o ambas (INOUT).
- Ejemplo de llamada: CALL procedure(param1, param2).

```
DELIMITER //
CREATE PROCEDURE nombre_procedimiento ([IN | OUT | INOUT] parametro1 tipo_dato)
BEGIN
    --Sentencia SQL
END;
//
DELIMITER ;
```



Function

- Propósito: Similar a procedure, sin embargo, este debe retornar un valor, y se suele utilizar para cálculos que retornan solo un valor.
- Uso: Se puede invocar dentro de consultas SQL, por lo tanto es útil para consultas o cálculos repetitivos.

Características

- Debe retornar un valor usando RETURN.
- Se puede usar en un SELECT, WHERE y otras partes de consultas SQL.
- Solo tiene parámetros de entrada.

```sql
DELIMITER //
CREATE FUNCTION calcular_descuento(precio DECIMAL(10,2))
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    RETURN precio * 0.9;
END;
//
DELIMITER;
```

Calcula precio con descuento y lo retorna

Function Sintaxis

- CREATE FUNCTION <nombre>: Define y crea una nueva función.
- RETURNS: Determina el tipo de dato que se retornará.
- DETERMINISTIC | NOT DETERMINISTIC: Indica si para el mismo valor de entrada siempre retornará el mismo valor de salida. Una función NOT DETERMINISTIC sería, por ejemplo, una que retorna el tiempo actual, ya que varía según el tiempo y no la entrada.

Ejemplo de llamada: SELECT function(param1);

```
DELIMITER //
CREATE FUNCTION nombre_funcion (parametro tipo_dato, ...)
RETURNS tipo_dato_de_retorno
[DETERMINISTIC | NOT DETERMINISTIC]
BEGIN
    --  Sentencia SQL
    RETURN valor_de_retorno;
END;
//
DELIMITER ;
```

Trigger

- Propósito: Fragmento SQL que se ejecuta automáticamente en respuesta a algunos eventos (como INSERT, UPDATE o DELETE), esto en búsqueda de asegurar integridad de los datos y automatizar respuestas.
- Uso: Se activa automáticamente cuando ocurre un evento, sin necesidad de ser llamado. Se utiliza para validar datos, mantener un registro de auditoría o gestionar tareas automáticas.

Características

- No retorna ningún valor.
- No puede ser llamada, solo se activa en respuesta a algún evento.
- Puedes acceder a las filas afectadas usando NEW y OLD para ver estado actual y anterior de los datos.

```
DELIMITER //
CREATE TRIGGER auditoria_insercion_user
AFTER INSERT ON usuarios
FOR EACH ROW
BEGIN
    INSERT INTO auditoria(usuario_id, accion, fecha)
    VALUES (NEW.id, 'Insercion', NOW());
END;
//
DELIMITER;
```

Guarda datos de inserción con fecha en tabla aparte “auditoria”

Trigger Sintaxis

- CREATE TRIGGER <nombre>: Crea un nuevo Trigger.
- { Before , After}: Define si se ejecuta antes o después de la operación.
- { INSERT | UPDATE | DELETE }: Evento que inicia el Trigger.
- ON <nombre_tabla>: Tabla sobre la que se activa el trigger.

```
DELIMITER //
CREATE TRIGGER nombre_trigger
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE }
ON nombre_tabla
FOR EACH ROW
BEGIN
    -- Sentencias SQL que se ejecutan en el evento del trigger
END;
//
DELIMITER ;
```

Resumen

Objeto	Propósito	Invocación	Retorno
Procedure	Ejecutar tareas complejas en múltiples pasos.	CALL	Mediante parámetros OUT
Function	Realizar cálculos y retornar solo un valor	SELECT	Obligatorio con RETURN
Trigger	Ejecuta código automatico ante eventos	Activado automaticamente	No aplica

Elementos Generales

- DECLARE: Declara variables

```
DECLARE nombre_variable TIPO_DATO [DEFAULT valor];
```

- SET: Para asignar valores a variables

```
SET nombre_variable = valor;
```

- Estructuras Condicionales

```
IF condición THEN
    -- Código a ejecutar si la condición es verdadera
ELSE
    -- Código a ejecutar si la condición es falsa
END IF;
```

Elementos Generales

- CASE

```
CASE
    WHEN condición1 THEN
        -- Código a ejecutar si la condición1 es verdadera
    WHEN condición2 THEN
        -- Código a ejecutar si la condición2 es verdadera
    ELSE
        -- Código a ejecutar si ninguna condición es verdadera
END CASE;
```

- LOOP: Bucles, la condición de salida se determina con un IF

```
nombre_bucle: LOOP
    -- Código dentro del bucle
    IF condición_salida THEN
        LEAVE nombre_bucle;
    END IF;
END LOOP;
```

Elementos Generales

- SIGNAL: Señal de error hecha por programador

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Mensaje de error';
```

- Cursores: Recorren filas de tablas, se declara con una consulta SQL

```
DECLARE nombre_cursor CURSOR FOR
    SELECT columna1, columna2 FROM tabla;

OPEN nombre_cursor;
FETCH nombre_cursor INTO variable1, variable2;

CLOSE nombre_cursor;
```

Ejercicio

Tienes la siguiente base de datos

- **cliente**(IdCliente, nombre, direccion, telefono, ciudad)
- **producto**(IdProducto, descripcion, precio,costo)
- **venta**(IdProducto, IdCliente, cantidad, fecha, IdVenta)

Realiza las siguientes actividades:

1) Implementar un procedimiento que permita insertar n tuplas en la tabla Producto, con el formato $(i+k, \text{“PolyStation } i+k\text{”}, (i+k)*5000, (i+k)*2500)$ en donde k es el máximo id de la tabla, e $i = 1, ..., n$.

2) Implementar un procedimiento, el cual debe crear la tabla **Proveedores**(id int , ciudad varchar(15), descripcion varchar(50)). Este recibe, por input, un número **n** de tuplas a agregar utilizando la siguiente regla:

   - Si i mód 5 es 0 entonces agregar la tupla (i,“yavin i”,“un proveedor muy interesante”)
   - En todo otro caso, entonces agregar la tupla (i,“dude i”,“un proveedor amable”),

   en donde $i = 1, ...n$.

Ejercicio

Tienes la siguiente base de datos

- **cliente**(IdCliente, nombre, direccion, telefono, ciudad)
- **producto**(IdProducto, descripcion, precio,costo)
- **venta**(IdProducto, IdCliente, cantidad, fecha, IdVenta)

Realiza las siguientes actividades:

3) Implementar un procedimiento, el cual debe crear la tabla **premmy**(idPremmy int, descripcion varchar(100), precio int). Esta tabla debe llenarse con aquellas tuplas de Producto (descripción y precio), en donde el precio sea mayor a 100,000. Debe utilizar cursores.

4) Implementar un trigger, el cual cada vez que se inserta un nuevo ítem a la tabla producto, se debe verificar que el precio de venta sea el doble del costo. Además, en caso que la descripcion sea NULL, rellenarla con "Nombre de producto a rellenar por el administrador".