**Asignatura:** BASES DE DATOS AVANZADAS **Fecha:** 16 de mayo del 2018

**Estudiantes:** Nancy Paola Sánchez Roldan 20152778319 - Brayan Moreno Cupitra 20162578005

**QUIZ: PROCEDIMIENTOS ALMACENADOS Y FUNCIONES**

**Definir y realizar 5 ejemplos de: vistas, procedimientos almacenados y funciones y triggers**

**I. Definición de un procedimiento almacenado:** Conjunto de comandos que pueden ser ejecutados directamente en el servidor, es decir, será ejecutado por el servidor de Base de Datos y no por el programa cliente que lo accede, permitiendo la ejecución de una acción o conjunto de acciones específicas.

**Funciones de un procedimiento almacenado:**

Algunas situaciones en que los procedimientos almacenados pueden ser particularmente útiles:

* Cuando múltiples aplicaciones cliente se escriben en distintos lenguajes o funcionan en distintas plataformas, pero necesitan realizar la misma operación en la base de datos.
* Cuando la seguridad es muy importante. Los bancos, por ejemplo, usan procedimientos almacenados para todas las operaciones comunes. Esto proporciona un entorno seguro y consistente, y los procedimientos pueden asegurar que cada operación se loguea apropiadamente. En tal entorno, las aplicaciones y los usuarios no obtendrían ningún acceso directo a las tablas de la base de datos, sólo pueden ejecutar algunos procedimientos almacenados.

**Elementos de los procedimientos almacenados**

Los procedimientos almacenados están compuestos por algunos de estos elementos:

* Parámetros de entrada (pueden esperar parámetros)
* Parámetros de salida (pueden devolver resultados)
* Declaración de variables (puede usarse variables en su cuerpo)
* Cuerpo del procedimiento (en su cuerpo se indican las acciones a realizar)

Tanto los parámetros de entrada como los de salida son opcionales. Podemos tener un procedimiento que no tenga parámetros de entrada, pero sí de salida y viceversa. los parámetros de entrada sirven para pasarle datos al procedimiento que necesita para llevar a cabo una acción determinada. Los parámetros de salida devuelven al programa cliente que ejecutó el procedimiento el resultado del mismo.

**Sintaxis**

Para crear un procedimiento almacenado se debe emplear la sentencia CREATE PROCEDURE.

**CREATE PROCEDURE** <nombre\_procedure> [@param1 <tipo>, ...]

**AS**

-- Sentencias del procedure

Para modificar un procedimiento almacenado se debe emplear la sentencia ALTER PROCEDURE.

**ALTER PROCEDURE**<nombre\_procedure> [@param1 <tipo>, ...]

**AS**

-- Sentencias del procedure

Para ejecutar un procedimiento almacenado se debe emplear la sentencia EXECUTE (O "EXEC") seguido del nombre del procedimiento y un valor para el parámetro:

**EXEC** <nombre\_procedure> ‘param1, …’

**Ejemplo 1:**

El siguiente ejemplo muestra un procedimiento almacenado, llamado spu\_addCliente que añade clientes, o sea, inserta un registro en la tabla ‘Clientes’

**CREATE PROCEDURE** spu\_addCliente

@nombre **VARCHAR(100),**

@apellidos **VARCHAR(100),**

@f\_Naciento datetime,

@email **VARCHAR(50)**

**AS**

**INSERT INTO CLIENTES (**nombre, apellidos, f\_nacimiento, email)

**VALUES** (@nombre, @apellidos, @f\_naciento, @email)

**EXEC** spu\_addCliente ‘Fabian’,’Sambrano’, ’1996-05-01’, ‘fsambrano@gmail.com’

Ejemplo 2: Parque Natural

Mostrar Datos de un Investigador específico

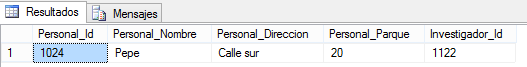
create procedure DatosInvestigador

@Identificacion int

AS

select Personal.\*, Investigador\_Id from Personal, Investigador where Investigador\_Personal = Personal\_Id and Personal\_Id = @Identificacion

exec DatosInvestigador 1024



Ejemplo 3: Parque Natural

Mostrar Datos de los Investigadores

create procedure PersonalInvestigador

as

select Personal.\* from Personal, Investigador where Personal\_Id=Investigador\_Personal;

exec PersonalInvestigador

Ejemplo 4: Parque Natural

Insertar datos de Especie.

create procedure InsertarEspecies

@id int,

@nombreV varchar(30),

@nombreC varchar(30),

@area varchar(30)

as

insert into Especie(Especie\_Id, Especie\_NombreVulgar, Especie\_NombreCientifico, Especie\_Area) values (@id, @nombreV, @nombreC, @area)

exec InsertarEspecies 1001, 'lagartera', 'Setaria verticillata', 50

exec InsertarEspecies 1002, 'juncia marina', 'Scirpus maritimus', 50

exec InsertarEspecies 1003, 'espiguilla', 'Poa annua', 20

exec InsertarEspecies 1004, 'vallico', 'Lolium rigidum', 20

exec InsertarEspecies 1005, 'zadorija', 'Hypecoum imberbe', 50

Ejemplo 5: Parque Natural

Buscar las especies que estén en un parque x

create procedure BuscarEspecieXParque

@parque varchar(40)

as

select Especie.\*, Parque\_Nombre from ParqueNatural, Area, Especie where Especie\_Area = Area\_Id and Area\_ParqueNatural = Parque\_Id and Parque\_Nombre = @parque

exec BuscarEspecieXParque 'parque'

**II. Definición de Función**

Una función es un conjunto de sentencias que operan como una unidad lógica.

La diferencia entre una función y un procedimiento es que la función devuelve valores. Estos valores pueden ser utilizados como argumentos para instrucciones SQL, tal como lo hacemos normalmente con otras funciones como son, por ejemplo, MAX() o COUNT().

**Elementos de una función**

Una función tiene:

* Un nombre,
* Retorna un parámetro de salida
* y opcionalmente acepta parámetros de entrada.

Las funciones de SQL Server no pueden ser modificadas, las funciones definidas por el usuario sí.

**Tipos de funciones**

SQL Server ofrece varios tipos de funciones para realizar distintas operaciones. Se pueden clasificar de la siguiente manera:

* **Funciones de agregado**

Realizan un cálculo sobre un conjunto de valores y devuelven un solo valor. Se pueden usar en la lista de selección o en la cláusula HAVING de una instrucción SELECT. Puede usar una agregación en combinación con la cláusula GROUP BY para calcular la agregación en las categorías de filas. Use la cláusula OVER para calcular la agregación en un intervalo de valor específico. La cláusula OVER no puede seguir las agregaciones GROUPING o GROUPING\_ID.

Estas son "count", "sum", "min" y "max".

* **Funciones analíticas**

calculan un valor agregado basándose en un grupo de filas. A diferencia de las funciones de agregado, estas funciones pueden devolver varias filas para cada grupo. Puede usar funciones analíticas para calcular medias móviles, totales acumulados, porcentajes o resultados de N valores superiores dentro de un grupo.

* **Funciones de categoría**

Devuelven un valor de categoría para cada fila de una partición. Según la función que se utilice, algunas filas pueden recibir el mismo valor que otras. Las funciones de categoría son no deterministas.

* **Funciones de conjuntos de filas**

Devuelven un objeto que se puede usar como referencias de tabla en una instrucción SQL.

* **Funciones escalares**

Operan sobre un valor y después devuelven otro valor. Las funciones escalares se pueden utilizar donde la expresión sea válida.

Las funciones escalares pueden agruparse de la siguiente manera:

- *De configuración*: retornan información referida a la configuración.

Ejemplo:

select @@version;

retorna la fecha, versión y tipo de procesador de SQL Server.

-*De cursores:* retornan información sobre el estado de un cursor.

- *De fecha y hora*: operan con valores "datetime" y "smalldatetime". Reciben un parámetro de tipo fecha y hora y retornan un valor de cadena, numérico o de fecha y hora.

- *Matemáticas:* realizan operaciones numéricas, geométricas y trigonométricas.

- *De metadatos*: informan sobre las bases de datos y los objetos.

- *De seguridad*: devuelven información referente a usuarios y funciones.

-*De cadena*: operan con valores "char", "varchar", "nchar", "nvarchar", "binary" y "varbinary" y devuelven un valor de cadena o numérico.

- *Del sistema*: informan sobre opciones, objetos y configuraciones del sistema. Ejemplo:

select user\_name();

- *Estadísticas del sistema*: retornan información referente al rendimiento del sistema.

- *Texto e imagen*: realizan operaciones con valor de entrada de tipo text o image y retornan información referente al mismo.

**Sintaxis de una función**

Para crear una función se debe emplear la sentencia CREATE FUNCTION

**CREATE** [OR REPLACE] **FUNCTION** [esquema].nombre-función

(nombre-parámetro {IN | OUT | IN OUT} tipo-de-dato, ...)

**RETURN** tipo-de-dato {IS | AS}

Declaración de variables;

Declaración de constantes;

Declaración de cursores;

**BEGIN**

Cuerpo del subprograma PL/SQL;

**EXCEPTION**

Bloque de excepciones PL/SQL;

**END;**

**Descripción de la sintaxis:**

* Nombre-parámetro: es el nombre que queramos dar al parámetro. Podemos utilizar múltiples parámetros. En caso de no necesitarlos, podemos omitir los paréntesis.
* IN: especifica que el parámetro es de entrada y que por tanto dicho parámetro tiene que tener un valor en el momento de llamar a la función o procedimiento. Si no se especifica nada, los parámetros son por defecto de tipo entrada.
* OUT: especifica que se trata de un parámetro de salida. Son parámetros cuyo valor es devuelto después de la ejecución el procedimiento al bloque PL/SQL que lo llamó. Las funciones PLSQL no admiten parámetros de salida.
* IN OUT: Son parámetros de entrada y salida a la vez.
* Tipo-de-dato: Indica el tipo de dato PLSQL que corresponde al parámetro (NUMBER, VARCHAR2, etc).

**Ejemplo 1: Creación de una función**

**CREATE OR REPLACE FUNCTION** obtener\_salario

(w\_código\_emp **IN** emp.código\_emp%TYPE)

**RETURN NUMBER**

**IS** w\_salario emp.salario\_emp%TYPE;

**BEGIN**

**SELECT** salario\_emp **INTO** w\_salario

**FROM** emp

**WHERE** código \_emp = w\_código\_emp;

**RETURN** w\_salario;

**END** obtener\_salario;

* Cada función **debe devolver** un valor del tipo especificado utilizando la **sentencia RETURN**

**Ejemplo 2: Mostrar historial de los vigilantes con fecha**

create function fechaLetras (@fecha as date)

returns varchar(30)

as

begin

declare @resultado varchar(30)

set @resultado = CONCAT(day(@fecha), ' de ', datename(month,@fecha), ' del ', year(@fecha))

return @resultado

end

select dbo.fechaLetras('2018-05-17') as [fecha Actual]

go

select \* from Historial

select Historial\_Id, Historial\_Vigilante, Historial\_Area, Historial\_Estado, dbo.fechaLetras(Historial\_Fecha) as [Fecha] from Historial

**III. Definición de Trigger**

Un Trigger, también llamado Disparador, en una base de datos, es un procedimiento que se ejecuta cuando se cumple una condición establecida que está asociado con las tablas de dicha base de datos.

**Funcionalidad de un Trigger**

Los triggers son usados para mejorar la administración de la Base de Datos, sin necesidad de que el usuario ejecute sentencias SQL y pueden ser extremadamente útiles para llevar a cabo los siguientes casos:

* Cuando los datos de una tabla son generados desde otro tipo de procesos y es necesario que se controlen los valores, para evitar que algunos campos tomen valores concretos.
* Para duplicar los contenidos de una tabla automáticamente y en tiempo real.
* Para implementar restricciones sobre valores que pueden tomar ciertos campos de la tabla.
* Incrementar automáticamente los valores de un campo.
* Para controlar las modificaciones de los valores de los campos.
* Para realizar actualizaciones de una tabla en cascada.
* Para modificar campos o registros en una tabla que un usuario no puede modificar directamente.

**Elementos de un Trigger**

La estructura básica de un trigger es:

* *Llamada de activación***:** es la sentencia que permite "disparar" el código a ejecutar.
* *Restricción:* es la condición necesaria para realizar el código. Esta restricción puede ser de tipo condicional o de tipo nulidad.
* *Acción a ejecutar:* es la secuencia de instrucciones a ejecutar una vez que se han cumplido las condiciones iniciales.

**Sintaxis para la creación y ejecución de un trigger**

**CREATE TRIGGER** <trigger name> <BEFORE|AFTER> <INSERT|DELETE|UPDATE>

**ON** <relation name> **FOR EACH** <ROW|STATEMENT>

**EXECUTE PROCEDURE** <procedure name> (<function args>);

* El nombre del trigger se usará si se desea eliminar el trigger. Se usa como argumento del comando **DROP TRIGGER.**
* La palabra siguiente determina si la función debe ser llamada antes (BEFORE) o después (AFTER) del evento.
* El siguiente elemento del comando determina en que evento/s será llamada la función. Es posible especificar múltiples eventos utilizado el **operador OR.**
* El nombre de la relación (relation name) determinará la tabla afectada por el evento.
* La instrucción FOR EACH determina si el trigger se ejecutará para cada fila afectada o bien antes (o después) de que la secuencia se haya completado.
* El nombre del procedimiento (procedure name) es la función C llamada.
* Los argumentos son pasados a la función en la estructura CurrentTriggerData. El propósito de pasar los argumentos a la función es permitir a triggers diferentes con requisitos similares llamar a la misma función.

Si una función trigger ejecuta consultas SQL (utilizando SPI) entonces estas funciones pueden disparar nuevos triggers. Esto es conocido como triggers en cascada. No hay ninguna limitación explicita en cuanto al número de niveles de cascada.

**Sintaxis para la creación de un trigger en SQL**

**CREATE** [OR REPLACE] **TRIGGER** nombre

{**BEFORE | AFTER | INSTEAD OF**} {**INSERT | UPDATE | DELETE**} [of columna, ...]

**ON** tabla

[FOR EACH ROW]

[**WHEN** (condiciones)]

[declare

-- Declaraciones del Trigger

...

]

**BEGIN**

-- Código del Trigger

...

**END;**

**Tipos de Triggers**

Existen dos tipos de disparadores que se clasifican según la cantidad de ejecuciones a realizar:

* **Row Triggers** (o Disparadores de fila): son aquellas que se ejecutaran cada vez que se llama al disparador desde la tabla asociada al trigger
* **Statement Triggers** (o Disparadores de secuencia): son aquellos que sin importar la cantidad de veces que se cumpla con la condición, su ejecución es única.

**Ejemplo 1:**

Si se inserta o actualiza el registro de un empleado de manera que el salario sea superior a 5000 euros, el trigger actualizará el salario a 5000 euros independientemente del valor que se haya insertado o introducido al modificar el registro.

**CREATE TRIGGER** tr1\_empleados

**BEFORE INSERT OR UPDATE** OF salario

**ON** empleados

**FOR EACH ROW**

**WHEN** (:new.salario > 5000);

**BEGIN**

**UPDATE** empleados

**SET** salario = 5000

**WHERE** empleado\_id = :new.empleado\_id;

**END;**

**Ejemplo 2:**

Si se inserta un animal muestra el siguiente mensaje ‘Animal Registrado’

create trigger NuevoAnimal

on Animal

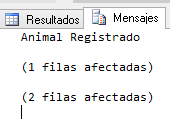
for insert

as

Print 'Animal Registrado'

go

insert into Animal(Animal\_Id, Animal\_Nombre, Animal\_Edad, Animal\_Especie) values (1, 'Leon', 3, 1001)

**Ejemplo 3:**

Si se inserta un animal muestra el siguiente mensaje ‘Animal Registrado’

create trigger NuevoAnimal

on Animal

for insert

as

Print 'Animal Registrado'

go

insert into Animal(Animal\_Id, Animal\_Nombre, Animal\_Edad, Animal\_Especie) values (1, 'Leon', 3, 1001)

**REFERENCIAS**

[1] Capítulo 19. Procedimientos almacenados y funciones. (s.f). n/a. Recuperado de: https://manuales.guebs.com/mysql-5.0/stored-procedures.html

[2] Procedimientos almacenados. (s.f). ECURED Conocimiento con todos y para todos. Recuperado de: https://www.ecured.cu/Procedimientos\_almacenados

[3] 24 – Funciones. (s.f). n/a. http://www.tutorialesprogramacionya.com/sqlserverya/temarios/descripcion.php?cod=30&punto=24&inicio=

[4] Microsoft. (2017) ¿Cuáles son las funciones de base de datos SQL? Recuperado de: https://docs.microsoft.com/es-es/sql/t-sql/functions/functions?view=sql-server-2017

[5] Aguilar. J. (2015). Los Trigger en una base de datos. Recuperado de: https://www.somosbinarios.es/los-trigger-una-base-datos/

[6] A2SISTEMAS 2012. (s.f) ¿Que son los triggers? Recuperado de: http://www.a2sistemas.com/blog/2009/01/09/¿que-son-los-triggers/

[7] PostgreSQL. (s.f). n/a. Triggers (disparadores). Recuperado de: http://es.tldp.org/Postgresql-es/web/navegable/todopostgresql/triggers.html

[8] Wikipedia. (s.f). Trigger (base de datos). Recuperado de: https://es.wikipedia.org/wiki/Trigger\_(base\_de\_datos)#Componentes\_principales

[9] Grupo de ingeniería del Software y Bases de Datos. (2011). Funciones, procedimientos, secuencias y cursores en Oracle. Universidad de Sevilla. Recuperado de: http://www.lsi.us.es/docencia/get.php?id=5784