

Procedimiento almacenado (Store Procedure)

Sitio: Agencia de Habilidades para el Futuro
Curso: Administración de Base de Datos 1° D
Libro: Procedimiento almacenado (Store Procedure)

Imprimido por: RODRIGO PINTO
Día: miércoles, 27 de noviembre de 2024, 09:28

Tabla de contenidos

1. Procedimientos almacenados

1.1. Variables de usuario

1.2. Delimitador

1.3. Procedure / Function

1.4. Estructuras de control

2. Parámetros

2.1. IN (Entrada)

2.2. OUT (Salida)

2.3. INOUT (Entrada - Salida)

Procedimientos almacenados: conceptos básicos



¿Qué son?

Esta semana damos inicio al mundo de los procedimientos almacenados, seguramente lo que escuchaste sobre ellos es que son complicados y difíciles de comprender.

Pero..., si seguís la explicación, documentás lo que razonás junto a las sentencias lo vas a poder dominar.

Lo difícil no es la sintaxis, porque si se quiere evaluar dos condiciones o repetir líneas de código te fijas en el formato, lo difícil es analizar y razonar los pasos y lo que debe proyectar las consultas que contengan.



Comencemos, el **procedimiento** como idea general, se presenta como un **subalgoritmo** que forma parte del algoritmo principal, el cual permite resolver una tarea específica.

Son como conjuntos de instrucciones escritas en el **lenguaje SQL**. Su objetivo es realizar una tarea determinada, desde operaciones sencillas hasta tareas muy complejas.

Tienen las siguientes características:

- **Están dentro de la Base de datos.**
- **Son pequeños programas.**
- **Mejora la seguridad.**
- **Implementan funcionalidad.**



¿Por qué usar procedimientos?

Porque el código se **pre-compila**, se reduce el tráfico entre cliente y servidor y permite reutilización de código.



¿Por qué *no* usar procedimientos?

El lenguaje usado para escribirlos usualmente depende del sistema gestor de base de datos y cada sistema gestor tiene un grado distinto de sofisticación en estos lenguajes.

Para crear los procedimientos debemos conocer algunos aspectos:

- **Variables de usuario.**
- **Delimitador.**
- **Sentencias que se pueden usar dentro de un procedimiento.**
- **Parámetros.**

Variables de usuario



¿Qué son?

Una variable en programación es un espacio de memoria que almacena un dato de un determinado tipo, en donde el **contenido varía** durante la ejecución del programa.

Pero **¿Qué es una variable de usuario?** Cada vez que abrís el entorno para ejecutar una consulta, ingresas con un **usuario**. Si ingresás con el *Shell* lo hacés como **root**.

Ese usuario abre una sesión que se mantiene hasta cerrar el entorno, como lo muestra la siguiente imagen.

```
Administrador: XAMPP for Windows - mysql -u root -p

Setting environment for using XAMPP for Windows.
Sandra@DESKTOP-BAL3UTG c:\xampp
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 10.1.36-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Si ese usuario necesita almacenar **un valor** porque lo tiene que utilizar después, puede crear una **variable**, esa variable recibe el nombre de **variable de usuario**.

Se le debe asignar un nombre para que el sistema gestor puede identificarla entre sus posiciones de memoria. El nombre tiene que ser significativo, hace referencia a lo que contiene, y es recomendable que tenga como **máximo 8 caracteres**. Además del nombre para que sea reconocido como variable de usuario se **le antepone el símbolo arroba (@)**.

Los nombres de las variables son **case insensitive**, es decir, da igual si están escritas en **mayúsculas o minúsculas**:

Se puede acceder a esas variables en cualquier momento mientras no se cierre la conexión, lo que nos permite incluso hacer cualquier tipo de operación con ellas:

Por ejemplo, si quiero almacenar un nombre puede ser @Nom.

¿Cómo se escribe? De la siguiente manera: **SET @Nom = VALOR;**

```
MariaDB [(none)]> set @Nom = "Amalia";  
Query OK, 0 rows affected (0.00 sec)  
  
MariaDB [(none)]> select @Nom as NOMBRE;  
+-----+  
| NOMBRE |  
+-----+  
| Amalia |  
+-----+  
1 row in set (0.00 sec)  
  
MariaDB [(none)]>
```

La palabra reservada **set** asigna un valor a la variable

Para visualizar su contenido se proyecta con el **select**

Observar que se renombra la columna

Delimitador



¿Qué es?

El **delimitador** es el símbolo que se coloca al final de la instrucción para que el sistema gestor se entere que la sintaxis esta lista para ser ejecutada, SQL usa por defecto el **punto y coma** como delimitador.

Cuando trabajamos en un procedimiento se crea un **bloque con inicio y final**, dentro del bloque se escriben **sentencias SQL** y por ende cada una de ellas finaliza con el punto y coma, pero se genera un inconveniente.



El inconveniente es que al colocar el punto y coma **el sistema gestor ejecuta**, pero no es el final del bloque del procedimiento y esto genera un **error**.



¿Cómo lo solucionamos?

Lo que podemos hacer es **cambiar el delimitador** mientras escribimos el procedimiento para que desconozca el punto y coma de la sintaxis interna, es decir, la que pertenece al procedimiento y está entre el bloque de inicio y el final.

Para cambiar el delimitador se usa la palabra **reservada delimiter** y a continuación colocamos cualquier carácter. En general si lees bibliografía sobre el tema ves que el delimitador son **dos barras (//)**.



Veamos cómo funciona.

```

MariaDB [taller]>
MariaDB [taller]> /* =====
/*> usamos TALLER
/*> ===== */
MariaDB [taller]> use taller;
ERROR 1049 (42000): Unknown database 'taller;'
MariaDB [taller]>
MariaDB [taller]> /* =====
/*> CAMBIAMOS el DELIMITADOR
/*> ===== */
MariaDB [taller]>
MariaDB [taller]> delimiter //
MariaDB [taller]>
MariaDB [taller]> /* =====
/*> A partir de este momento DESCONOCE el ;
/*> y considera como DELIMITADOR a //
/*> ===== */
MariaDB [taller]>
MariaDB [taller]> select * from presup //
+-----+-----+-----+-----+-----+-----+
| NPresup | codF | fecha | DiagFinal | Monto | aceptado |
+-----+-----+-----+-----+-----+-----+
| 70100 | 8003 | 2023-03-30 | Se debe reemplazar el motor en su totalidad. Rotura del Block del motor | 150000 | 0 |
| 70101 | 8004 | 2023-04-04 | Reemplazo de la grasa de la caja de cambio, y reparacion de la barra | 8750 | 0 |
| 70102 | 8005 | 2023-04-05 | Sustitucion del tren delantero y arreglo de la parrilla frontal | 14000 | 0 |
| 70103 | 8006 | 2023-04-06 | Reseteo de la computadora interna del vehiculo con programa oficial | 18500 | 0 |
| 70104 | 8007 | 2023-04-06 | Reparacion de cierre de puerta trasera izquierda y reemplazo de puerta derecha | 9700 | 0 |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

Usamos la Base **Taller**

Cambiamos el **Delimitador**

Ejecutamos con el nuevo **Delimitador**

Procedure/Function



Sentencias y procedimientos ¿Cuándo y cómo se usan?

- Las sentencias que podemos usar dentro de los procedimientos son: *insert, update, select, drop, create*.
- Las sentencias que **no** podemos usar dentro de los procedimientos son: *create procedure, alter procedure, drop procedure, create function, drop function, create trigger, drop trigger*.



¿Qué tipos de procedimientos podemos crear?

Podemos crear *procedure* y *function*.

¿Cómo los creamos? ¿Cómo los invocamos?

- Paso 1: *Procedure*

```
delimiter //
```

```
create procedure Nombre_del_Procedimiento()
```

```
begin
```

```
end //
```

Después de crear el "PROCEDIMIENTO" vemos dos palabras reservadas → **begin** y **end**

Inicio y fin

Ambas palabras forman el **bloque** del procedimiento
Las sentencias que se escriben dentro del bloque forman el cuerpo del procedimiento.

Cada una de ellas terminan con **punto y coma**

```
call Nombre_del_Procedimiento() //
```

Call → se utiliza para invocar al procedimiento

- Paso 2: *Function*

```
delimiter //

create function Nombre_de_la_Funcion returns << Tipo de dato >>

begin

    return << valor >>

end //

select Nombre_de_la_Funcion() //
```

La función **SIEMPRE** retorna un valor.
El valor de retorno es del tipo de dato declarado en el **create**

Antes del **end** aparece la palabra reservada **return** que permite la salida del valor que calcula

Select → se utiliza para invocar a la función



¿Cual es el código?

Principalmente el código que escribimos se refiere a la lógica que vamos a utilizar.

Cuando codificamos seguramente usamos variables, a diferencia de las variables de usuario estas se deben declarar y **su valor se mantiene mientras se ejecuta el procedimiento**; una vez que finaliza pierde su valor y no es reconocida por el sistema gestor.

```
/* el nombre debe ser significativo */  
  
DECLARE Nombre_Variable << Tipo de dato >>;  
  
/* se le asigna un valor con el SET */  
  
set Nombre_Variable = Valor;
```

Estructuras de control: *IF/WHILE/LOOP*



¿Para que son necesarias?

Necesitamos estructuras de control para tomar caminos o repetir líneas de código.



Vemos primero el bloque "Bloque IF"

Bloque **IF**

```
IF expresión THEN
```

```
    /*que hacer en caso correcto*/
```

```
ELSE
```

```
    /* que hacer en caso contrario */
```

```
END IF; /* necesario para cerrar el bloque */
```

La expresión es la condición que se evalúa, para luego tomar el camino en función del resultado.



Ciclos *loop While*

El ciclo *loop while*, prevalida la **expresión**, ¿Qué significa? **Mientras** la expresión es **verdadera ingresa** al ciclo.

```
Ciclos/loop WHILE
```

```
    WHILE expresión DO
```

```
        /*contenido del bucle*/
```

```
    END WHILE;
```



¿Cómo funciona?

- Entrá al ciclo.
- ejecutá línea a línea las sentencias.
- cuando llegás al **end while** regresá a evaluar la expresión y
- comienza todo nuevamente.

¿Cuándo sale? Cuando la expresión **no** se cumple.

Este ciclo permite repetir líneas de código.



Ciclo loop Etiqueta : *LOOP*

Como vemos **etiqueta es una cadena de caracteres** cualquiera. Para el caso de este bucle, su código se ejecutará hasta que se encuentre la sentencia **LEAVE ETIQUETA** cuyo objetivo es similar a la función que **cumple break** en algunos lenguajes de programación.

```
Ciclo/loop Etiqueta : LOOP

    ETIQUETA : LOOP

        /* Código del bucle*/

        IF variable >= VALOR THEN

            LEAVE ETIQUETA; /*se ejecuta hasta encontrar esta escritura*/

        END IF;

    END LOOP;
```

Observar que el código del bucle se ejecuta por lo menos **una vez** porque el **if** se coloca después si la condición se cumple ingresa al bloque **if**, se encuentra **leave etiqueta** y sale por el **end loop**.

Parámetros



¿Qué son?

Un **parámetro** representa **un valor que el procedimiento** espera que se pase al llamarlo. La declaración del procedimiento define sus parámetros. Cuando se define un procedimiento o una función, se especifica una lista de parámetros entre paréntesis inmediatamente después del nombre del procedimiento.

En el **store procedure** se puede usar parámetros de entrada, parámetros de salida y parámetros de **entrada / salida**. Reciben el nombre de **in, out, inout**.

- Los parámetros de entrada (**in**) son valores que toman esas variables y que no se conservarán una vez termine de ejecutarse el procedimiento. Es opcional colocar la palabra (**in**) ya que si no se especifica y el procedimiento presenta parámetros el sistema gestor lo considera de entrada.
- Los parámetros de salida (**out**) son valores que se establecen dentro del procedimiento y pueden ser accedidos más adelante.
- Los parámetros de entrada / salida (**inout**) es una combinación de las dos anteriores. La variable puede recibir valores de entrada y puede ser accedida más adelante.



En los siguientes capítulos veremos algunos ejemplos.

Para la explicación práctica **tomaremos como ejemplo la base de datos que usamos en la semana 7-Base taller.**

IN (Entrada)

Retomando el ejercicio de la semana 7-Base taller.

Construí un procedimiento que al ingresar el código de un repuesto (*Base Taller*) muestre el nombre, precio y cantidad disponible.

Armá la consulta que proyecte los datos solicitados usando la tabla repuesto y colocando como condición que el código sea el solicitado, como lo muestra la siguiente imagen.

```
delimiter //
```

```
create procedure Muestra(in codigo int)
```

```
begin
```

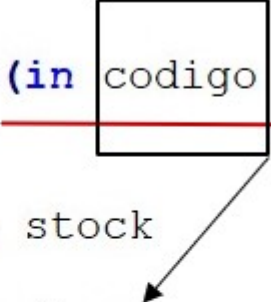
```
    select nombre, precio, stock
```

```
    from repuesto
```

```
    where codrep = codigo;
```

```
end//
```

```
call Muestra(123) //
```



Llamada con un valor que existe en la tabla

Esta **llamada** muestra la proyección de la consulta que esta en el procedimiento.

OUT (Salida)

Construí un procedimiento que al ingresar un valor devuelva en una variable la cantidad de repuestos que tienen un precio mayor al valor ingresado. Observá la siguiente imagen para ver como debe quedar:

```
delimiter //
```

```
create procedure Cuantos(in valor float, out contador int)
```

```
begin
```

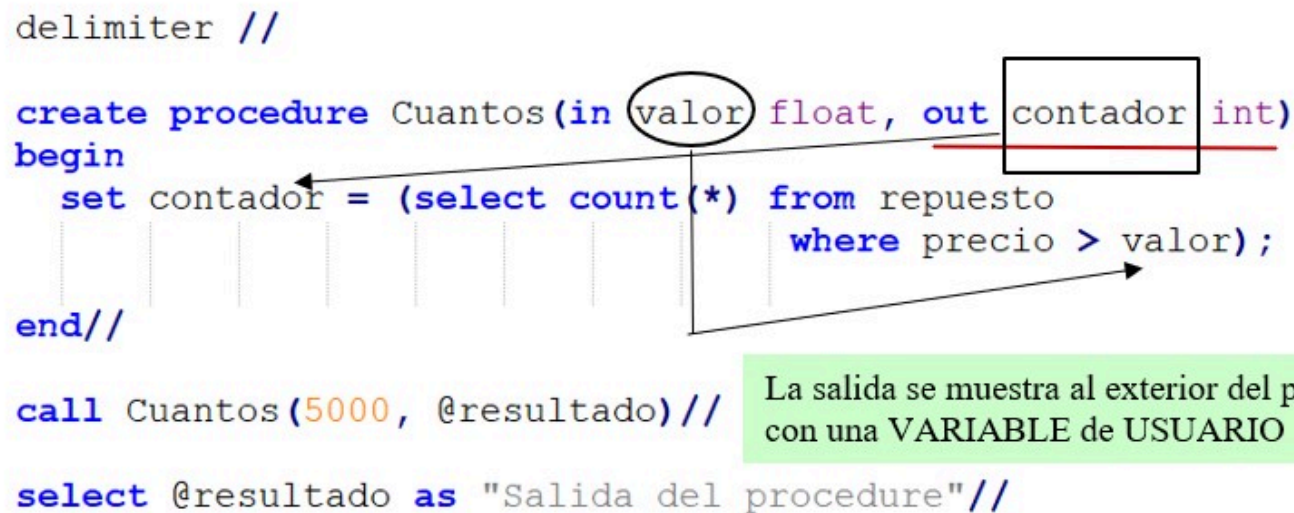
```
    set contador = (select count(*) from repuesto
```

```
                    where precio > valor);
```

```
end//
```

```
call Cuantos(5000, @resultado)//
```

```
select @resultado as "Salida del procedure"//
```



La salida se muestra al exterior del procedure con una VARIABLE de USUARIO

El contenido se visualiza con un SELECT


INOUT(Entrada - Salida)

Utilizá en un procedimiento una variable con valor previo e incrementale 10 unidades, devolviendo la misma variable.

Observá la imagen para comprender mejor el ejemplo.

```
delimiter //
```

```
create procedure ejemplo(inout resultado int)
begin
    set resultado = resultado + 10;
end //
```



Resultado es un acumulador, aparece en ambos lados de la igualdad

```
set @VarInOut = 123//
call ejemplo(@VarInOut) //
```

```
select @VarInOut//
```

Antes de la invocación al procedure se da un valor a la variable