Bases de Datos

Unidad 8:

Programación de bases de datos

Sesión 2

3.- Desarrollo de procedimientos almacenados

Un procedimiento o PROCEDURE es una rutina formada por un conjunto de instrucciones SQL y:

- Tiene un determinado **nombre** formado por una combinación de caracteres alfanuméricos y cualquiera de estos (. $ _).
- Puede recibir valores al ser llamado a ejecución a través de **parámetros** encerrados entre paréntesis.
- Puede devolver valores a través de **parámetros** encerrados entre paréntesis.
- Al desarrollar un procedimiento tenemos que declarar los parámetros que usará, que podrán ser de tres tipos: **IN, OUT , INOUT.**
- Un procedimiento se crea con la instrucción **CREATE PROCEDURE.**
- Un procedimiento se ejecuta con la instrucción **CALL nombreProc (params)**

- **Todo procedimiento queda asociado a la base de datos abierta cuando se creó el procedimiento.**
- Al ejecutar un procedimiento, el servidor MySQL ejecutará automáticamente una instrucción **USE basedatos**, donde **basedatos** es la asociada al procedimiento.
- De esta forma, podemos ejecutar un procedimiento asociado a una base de datos distinta a la que tenemos abierta especificando, en la llamada al procedimiento, un **cualificador** de la base de datos, de la forma:

CALL nomBASEDATOS.nomProc (parámetros).

- Si ejecutamos

CALL nomProc(parámetros),

será necesario que el procedimiento se encuentre en la base de datos que tengamos abierta.

3.- Desarrollo de procedimientos almacenados

- Cuando se crea un procedimiento, el servidor MySQL nos devolverá indicaciones sobre los errores que pueda tener o no el procedimiento. Si la sintaxis del procedimiento es correcta, el servidor almacenará dicho procedimiento, pero **no lo ejecutará en ese momento.**
- Si se intenta crear un procedimiento con un nombre que ya existe, el servidor MySQL no lo permite.

Sintaxis para crear un procedimiento:

CREATE PROCEDURE ***NomProc*** **([*parámetro1*[,...]])**
[*característica* ...]
BEGIN
Cuerpo_procedimiento
END

Elementos de la sintaxis de la instrucción CREATE PROCEDURE

Parámetro tiene la sintaxis:

[IN | OUT | INOUT] ***NomParam tipo***

tipo**:**

Cualquier tipo de dato MySQL

característica**:**

LANGUAGE SQL | [NOT] DETERMINISTIC | SQL SECURITY {DEFINER | INVOKER}
| COMMENT '*string*'

cuerpo_procedimiento**:**

Instrucciones SQL para realizar la tarea.

3.- Desarrollo de procedimientos almacenados

LANGUAGE SQL: Indica que el procedimiento está escrito en lenguaje SQL.

[NOT] DETERMINISTIC: Para indicar si el procedimiento es o no determinista.

¿Qué significa que una función o procedimiento es determinista?

CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA

CONTAINS SQL: Indica que no se realiza lectura o escritura de información por medio de sentencias SQL.

NO SQL: Indica que la rutina continue instrucciones NO SQL.

READS SQL DATA: Indica que la rutina lee información obtenida con instrucciones SQL (por ejemplo mediante SELECT) pero no escribe la información.

MODIFIES SQL DATA: Indica que la rutina podría escribir información por medio de instrucciones SQL (por ejemplo mediante INSERT o DELETE).

SQL SECURITY { DEFINER | INVOKER }: Determina los privilegios con los que se ejecuta la rutina. Pueden ser los de la cuenta que crea la rutina o los del usuario que la invoca.

COMMENT: Se utiliza para describir la rutina almacenada en el procedimiento y se muestra por medio del siguiente comando: **SHOW CREATE PROCEDURE statements**.

El carácter delimitador de final de instrucciones

- El delimitador de final de instrucciones de SQL es el punto y coma (;).
- Las instrucciones del cuerpo de un procedimiento deben terminar con punto y coma.
- Si mantenemos el delimitador punto y coma, se ejecutarían las instrucciones mientras se intenta crear el procedimiento y, éste no se crearía.
- Para poder crear procedimientos, tendremos que cambiar temporalmente, antes de empezar a crearlos, el carácter delimitador o finalizador de instrucciones SQL en MySQL.
- Para cambiar el carácter delimitador se usa la instrucción **DELIMITER**. Por ejemplo, para hacer que el delimitador de instrucciones sea '//', habrá que ejecutar:

  DELIMITER //

3.- Desarrollo de procedimientos almacenados

Ejemplo 1 de creación y ejecución de un procedimiento

```
delimiter //
```

```
CREATE  PROCEDURE listados()
BEGIN
        SELECT * FROM clientes;
        SELECT * FROM automoviles;
END//
```

```
call listados()//
```

O bien

```
delimiter ;
call listados();
```

3.- Desarrollo de procedimientos almacenados

Ejemplo 2: Crear y ejecutar un procedimiento **numcontratos** que recibe en un parámetro de entrada la matrícula de un coche y, a continuación, muestra las características del coche y devuelve en un parámetro de salida el número de contratos realizados sobre ese coche

```
CREATE  PROCEDURE numcontratos(m CHAR(7), OUT c INT)
BEGIN
        SELECT * FROM automoviles WHERE matricula=m;
        SELECT count(*) INTO c FROM contratos WHERE matricula=m;
END//
```

```
CALL numcontratos('3273BGH', @Num)//
```

```
SELECT @Num//
```

Uso de los parámetros en los procedimientos:

Los parámetros declarados en un procedimiento se pueden usar dentro del procedimiento.

Si un parámetro ha sido declarado IN, no se le puede asignar un valor dentro del procedimiento, aunque si que se podría consultar su valor. Si tenemos:

CREATE PROCEDURE ejemplo (IN num INT)

No podríamos usar esta instrucción dentro del procedimiento:

SELECT count(*) INTO num FROM contratos;

Si un parámetro es declarado OUT, no se puede usar ese parámetro para consultar su valor, si para modificarlo. Si tenemos:

CREATE PROCEDURE ejemplo (OUT num INT)

No podríamos usar esta instrucción dentro del procedimiento:

SELECT count(*) FROM contratos WHERE numcontrato=num;

En cambio, un parámetro INOUT podríamos usarlo tanto para lectura como para escritura.

Variables locales:

Además de los parámetros, en un procedimiento podemos declarar y usar **variables locales.**

Estas variables locales sólo tienen existencia mientras se ejecuta el procedimiento, después quedan destruidas.

Al igual que las demás instrucciones del procedimiento, la declaración de variables debe estar dentro del bloque BEGIN .... END. Para definir o declarar cualquier variable se usa la instrucción:

DECLARE nombre tipo[DEFAULT valor];

Donde tipo es cualquiera de los tipos admitidos por MySQL. Para modificar el valor de una variable o de un parámetro con el operador de asignación =, debe usarse la instrucción:

SET variable=expresión;

.

Ejemplo 3 de uso de variables locales: Crear un procedimiento **usovariable** que lista los vehículos con menos de 2500 kilómetros y, después, los vehículos con menos kilómetros que los anteriores más 5000 kilómetros.

```
CREATE  PROCEDURE usovariable()
 BEGIN
        DECLARE a INT;
        SET a=2500;
        SELECT * FROM automoviles WHERE kilometros<a;
        SET a=a+5000;
        SELECT * FROM automoviles WHERE kilometros<a;
 END//
```

Ejemplo 4:

Realiza un procedimiento que recibe la matrícula de un automóvil y muestra:

- *La marca y modelo del automóvil.*
- *El número de contratos de alquiler realizados para el automóvil.*
- *El número de clientes que han alquilado ese automóvil.*
- *Los nombres de los usuarios que han alquilado ese automóvil.*

```
CREATE PROCEDURE ejemplo4(IN mat char(7))
BEGIN
        SELECT marca,modelo FROM automoviles WHERE matricula=mat;
        SELECT count(*) FROM contratos WHERE matricula=mat;
        SELECT count(DISTINCT dnicliente) FROM contratos WHERE matricula=mat;
        SELECT DISTINCT nombre,apellidos FROM clientes INNER JOIN contratos ON
dnicliente = dni WHERE matricula=mat;
END
```

¿Qué falta al procedimiento anterior para estar "completo"?