TEMA 11: PROCEDIMIENTOS Y FUNCIONES

1. Procedimientos.

Los procedimientos y las funciones son rutinas o subprogramas. Un subprograma o rutina es un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez que se crea un subprograma, los clientes no necesitan ejecutar comandos individuales, sino que en su lugar pueden realizar una llamada al subprograma almacenado.

Los subprogramas pueden mejorar el rendimiento, ya que se necesita enviar menos información entre el servidor y el cliente.

Una función es un subprograma almacenado que devuelve un valor, mientras que los procedimientos no devuelven un valor.

Todo subprograma, como en todos los lenguajes de programación, consta de una cabecera y un cuerpo:

- La cabecera del subprograma incluirá:
  - El nombre del subprograma.
  - Los datos que recibe el subprograma, que reciben el nombre de parámetros. Por cada uno de ellos se indicará el nombre y el tipo de dato asociado.
  - El tipo del valor que devuelve, en el caso de que el subprograma sea una función.
- El cuerpo del subprograma, que incluirá:
  - Una sección de declaraciones, que es opcional.
  - Una sección de instrucciones, que incluirá en el caso de tratarse de una función, una instrucción para devolver el valor de retorno.

La instrucción para crear un procedimiento presenta la siguiente sintaxis:

```
CREATE PROCEDURE Nombre_Procedimiento ([Lista_Parámetros])
[características]
BEGIN
[DECLARE declaración1;
DECLARE declaración2;
…]
Instrucciones
END;
```

A todo procedimiento hay que asignarle un nombre y se debe especificar obligatoriamente una lista de parámetros o datos que recibe el subprograma. Puede ocurrir que un subprograma no reciba parámetros, en cuyo caso se pondrán los paréntesis de apertura y de cierre. La lista de parámetros tendrá el siguiente formato:

(Nombre_Parámetro$_1$ tipo$_1$, Nombre_Parámetro$_2$ tipo$_2$, …)

Por cada parámetro se debe indicar en orden el nombre del parámetro y su tipo y se separa la información referida a un parámetro de la información referida al siguiente parámetro por el símbolo coma (,).

A modo de ejemplo, se va a crear un procedimiento que muestra el mensaje “Hola mundo”. Hemos de tener en cuenta que el delimitador que debemos emplear dentro del cuerpo del procedimiento para separar una instrucción de la siguiente es el punto y coma (;), por lo que para delimitar el fin de la instrucción de creación del procedimiento deberemos emplear otro símbolo, por ejemplo //. Para indicar este hecho pondremos una primera instrucción DELIMITER //.

```
mysql> delimiter //
mysql> create procedure HolaMundo()
    -> begin
    -> select '¡Hola, mundo!' Mensaje;
    -> end;//
Query OK, 0 rows affected (0.00 sec)
```

Si al crear el procedimiento hubiésemos cometido algún error sintáctico, nos aparecería un mensaje de error y no se crearía. Para visualizar los errores cometidos en el procedimiento se puede usar la orden SHOW ERRORS.

Pues bien, una vez creado el procedimiento, para ejecutarlo tendremos que invocarlo o llamarlo, para lo que habrá que utilizar el comando CALL y escribir a continuación el nombre del procedimiento y los parámetros que le deseamos pasar (en este caso ninguno, pues no admite parámetros).

```
mysql> call HolaMundo();//
+---------------+
| Mensaje       |
+---------------+
| ¡Hola, mundo! |
+---------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.08 sec)
```

2. Variables.

Una variable sirve para almacenar información cuyo valor puede variar a lo largo de la ejecución del programa. Toda variable tiene asociado un identificador y un tipo de dato y se almacena en memoria principal.

Antes de emplear una variable en un subprograma es necesario declararla empleando la siguiente sintaxis:

```
DECLARE Identificador Tipo [NOT NULL] [DEFAULT Valor];
```

Se debe indicar en primer lugar el nombre de la variable o su identificador y a continuación su tipo de dato. Además, se puede incluir la cláusula NOT NULL, que indica que dicha variable no puede tomar valor nulo. También se puede asignar un valor por defecto a la variable escribiendo DEFAULT y a continuación el valor que se desee. Si se especifica NOT NULL, es obligatorio asignarle a la variable un valor por defecto.

Por ejemplo, a continuación se declara una variable con el identificador *codarti* como una cadena de longitud fija de 5 caracteres, y otra, *desarti*, como una cadena de caracteres de longitud variable con una longitud máxima de 30 caracteres.

```
DECLARE codarti CHAR(5);
DECLARE desarti VARCHAR(30);
```

A continuación vamos a declarar una variable numérica real *pvparti* con valor inicial 5 €, y otra, *cantarti*, como variable numérica entera que no puede tomar valor nulo y con valor inicial 1.

```
DECLARE pvparti float DEFAULT 5.0;
DECLARE cantarti int NOT NULL DEFAULT 1;
```

3. Operadores.

En los procedimientos y funciones creados en MySQL se puede emplear cualquiera de los operadores del LMD SQL y además el operador = combinado con la instrucción SET, lo cual permite asignar un valor a una variable. La sintaxis es la siguiente:

```
SET Nombre_variable = Valor;
```

Por ejemplo, para asignar a las variables *codarti* y *pvparti* los valores ‘A0001’ y 15.2 respectivamente, tendríamos que escribir:

```
SET codarti = 'A0001';
SET pvparti = 15.2;
```

4. Funciones

Una función, como se indicó anteriormente, es un subprograma que devuelve un dato. Pues bien, la instrucción para crear una función presenta la siguiente sintaxis:

```
CREATE FUNCTION Nombre_Función ([Lista_Parámetros]) RETURNS tipo
características
BEGIN
[DECLARE declaración1;
DECLARE declaración2;
…]
Instrucciones
END;
```

La sintaxis es como la de creación de procedimientos con las siguientes diferencias:

- Se debe escribir CREATE FUNCTION en lugar de CREATE PROCEDURE.
- Como las funciones devuelven un valor, después de la lista de parámetros se debe escribir obligatoriamente RETURNS y el tipo de dato del valor devuelto por la función.
- Para que las funciones se consideren seguras es preciso especificar al menos una de las siguientes características:
  - DETERMINISTIC: Indica que la función siempre produce los mismos resultados para los mismos parámetros de entrada.
  - NO SQL: Indica que la función no contiene comandos SQL.
  - READS SQL DATA: Indica que en la función hay sentencias SQL de lectura de datos (consultas), pero no sentencias SQL que modifican datos (*insert*, *update* y/o *delete*).

  Si una función no contiene ninguna de estas características en su cabecera, se considerará no segura y se producirá un error al intentar crearla.
- En el conjunto de instrucciones del cuerpo de la función se debe incluir una instrucción RETURN (expresión); o RETURN expresión; para devolver un valor.

A modo de ejemplo, se va a crear a continuación una función que reciba el código de un artículo y nos devuelva su descripción. Daremos al parámetro el nombre *cod* y pondremos como tipo char(5), que es el mismo tipo de dato que el atributo *CodArt* de la tabla *Articulo*. Además, como la función nos devuelve la descripción de un artículo, que es de tipo varchar(30) (tipo de dato del atributo *DesArt* de la tabla *Articulo*), pondremos returns varchar(30). Por otro lado, hemos de tener en cuenta que las sentencias SELECT incluidas dentro de subprogramas pueden incluir antes de la cláusula FROM la cláusula INTO seguida del nombre de una o varias variables para guardar el resultado de la consulta. Por ejemplo,

con las siguientes instrucciones se almacenaría en la variable *descri* la descripción del artículo con código A0043. La variable *descri*, como es normal, hay que declararla con anterioridad.

```
BEGIN
DECLARE descri varchar(30);
SELECT DesArt INTO descri FROM Articulo
WHERE CodArt = 'A0043';
…
```

Pues bien, como vemos, hay que declarar una variable para almacenar la descripción del artículo que luego devolverá la función. El código de la función será el siguiente:

```
mysql> use Pedidos
Database changed
mysql> create function LeerDescriArti (cod char(5)) returns varchar(30) reads sql data
    -> begin
    -> declare descri varchar(30);
    -> select DesArt into descri from Articulo
    -> where CodArt = cod;
    -> return descri;
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

La llamada a una función no se puede realizar con *call*, como se llama a los procedimientos, sino que lo que nos devuelve una función se puede poner en cualquier lugar en el que se puede poner una expresión, por ejemplo, en una sentencia SELECT que muestra información en pantalla. Así, por ejemplo para mostrar la descripción del artículo con código A0043, escribiremos:

```
mysql> select LeerDescriArti ('A0043') Descripción;//
+----------------+
| Descripción    |
+----------------+
| Bolígrafo azul |
+----------------+
1 row in set (0.00 sec)
```

Para eliminar un procedimiento o una función se emplea la orden DROP PROCEDURE o DROP FUNCTION respectivamente, cuyo formato es el siguiente:

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] Nombre_Subprograma
```

Así, para eliminar el procedimiento *HolaMundo* escribiremos:

```
mysql> drop procedure HolaMundo;
Query OK, 0 rows affected (0.01 sec)
```

5. Estructuras de control

Como en todos los lenguajes de programación, al escribir procedimientos y funciones en MySQL se pueden incluir diversas estructuras de control para controlar el flujo de ejecución de los programas.

Las estructuras de control se pueden emplear en MySQL englobadas dentro de subprogramas, a saber, procedimientos y funciones.

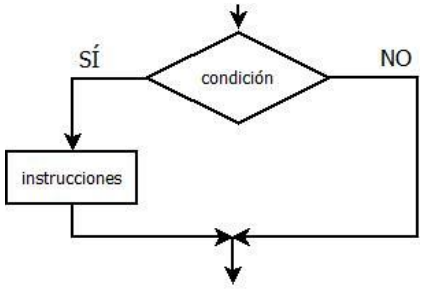
A continuación se van a estudiar las estructuras de control más relevantes que se pueden emplear en MySQL:

5.1. Estructura alternativa.

La estructura alternativa se construye mediante la sentencia IF y la sentencia CASE y permite decidir qué secuencia de código se va a ejecutar a continuación en función del cumplimiento o no de una determinada condición. En MySQL se pueden emplear varias estructuras alternativas, que se analizan a continuación:

- Estructura alternativa simple: En la estructura alternativa simple se comprueba una determinada condición y si se cumple, se ejecuta una secuencia de instrucciones; en caso contrario, dicha secuencia de instrucciones no se ejecuta. Su sintaxis es:

```
if condición then
   instrucción1;
   instrucción2;
   …
   instrucciónn;
end if;
```



Vamos a crear a continuación un procedimiento llamado *CambiarPrecioBoliRojo* que reciba un importe en euros y que compruebe el precio del artículo con descripción *Bolígrafo rojo normal* y si es mayor que 1 €, decremente su precio en el importe recibido como parámetro.

En este procedimiento, para mostrar un mensaje en pantalla que informe al usuario en caso de que se haya llevado a cabo la modificación, usaremos la función CONCAT que puede recibir varios parámetros entre paréntesis y separados por comas y se encarga de unir las cadenas de caracteres en el orden especificado.

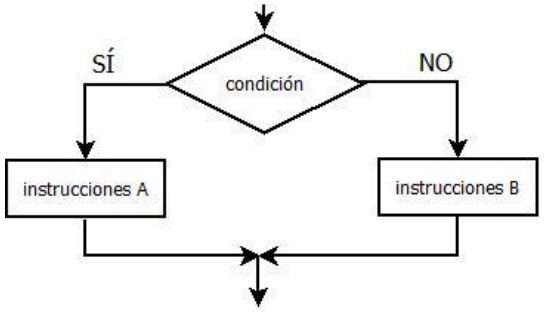
Lo que tenemos que hacer en nuestro procedimiento es seleccionar el precio del artículo con descripción 'Bolígrafo rojo normal' y almacenarlo en *pvp*. Luego hay que comprobar si el precio contenido en *pvp* es o no mayor que 1 € con una sentencia IF y en caso afirmativo, actualizar el precio y mostrar un mensaje indicativo por pantalla. Para ello, escribiremos el siguiente código:

```
mysql> create procedure CambiarPrecioBoliRojo (importe float)
    -> begin
    -> declare pvp float;
    -> select PVPArt into pvp from Articulo
    -> where DesArt = 'Bolígrafo rojo normal';
    -> if pvp>1 then
    ->    update Articulo set PVPArt = PVPArt - importe
    ->    where DesArt = 'Bolígrafo rojo normal';
    ->    select concat ('El bolígrafo rojo normal se ha abaratado ', importe, '
    ->                   euros') Mensaje;
    -> end if;
    -> end//
Query OK, 0 rows affected (0.09 sec)

mysql> call CambiarPrecioBoliRojo (0.06);//
+-----------------------------------------------------+
| Mensaje                                             |
+-----------------------------------------------------+
| El bolígrafo rojo normal se ha abaratado 0.06 euros |
+-----------------------------------------------------+
1 row in set (0.10 sec)

Query OK, 0 rows affected (0.13 sec)
```

El resultado de la ejecución, como se puede observar, es que se nos muestra el mensaje "`El bolígrafo rojo normal se ha abaratado 0.06 euros`" al tener este artículo un precio de 1,05 €, pasando su precio a ser 0,99 €.

- Estructura alternativa doble: en la estructura alternativa doble se comprueba una determinada condición; si se cumple, se ejecuta una secuencia de instrucciones y si no se cumple, otra secuencia de instrucciones. La sintaxis es la siguiente:

```
If condición then
   instrucción1;
   …
   instrucciónn;
else
   instrucciónn+1;
   …
   instrucciónn+m;
end if;
```



Vamos a crear un procedimiento que compruebe el precio del artículo cuyo código se pasa como parámetro, de manera que si es superior a 0,30 €, baje el precio en 0,02 €, y en caso contrario, lo suba en 0,02 €:

```
mysql> create procedure CambiarPrecio (codarti char(5))
    -> begin
    -> declare pvp float;
    -> select PVPArt into pvp from Articulo where CodArt=codarti;
    -> if pvp>0.3 then
    ->     update Articulo set PVPArt=PVPArt-0.02
    ->     where CodArt=codarti;
    ->     select concat ('El artículo con código ',codarti,
    ->                    ' se ha abaratado 0,02 euros') Mensaje;
    -> else
    ->     update Articulo set PVPArt=PVPArt+0.02
    ->     where CodArt=codarti;
    ->     select concat ('El artículo con código ', codarti,
    ->                    ' se ha encarecido 0,02 euros') Mensaje;
    -> end if;
    -> end//
Query OK, 0 rows affected (0.00 sec)
```

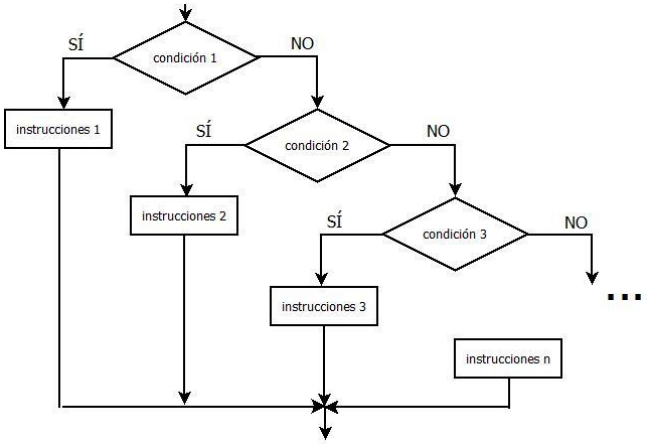
Vamos a llamar a este procedimiento para que modifique el precio del artículo con código A0089 (el sacapuntas).

```
mysql> call CambiarPrecio ('A0089');//
+-----------------------------------------------------------+
| Mensaje                                                   |
+-----------------------------------------------------------+
| El artículo con código A0089 se ha encarecido 0,02 euros |
+-----------------------------------------------------------+
1 row in set (0.07 sec)

Query OK, 0 rows affected (0.09 sec)
```

El resultado de la ejecución es el que se muestra porque el sacapuntas tenía un precio inferior a 0,30 €, concretamente 0,25 €.

- Estructura alternativa múltiple: la estructura alternativa múltiple consiste en varias estructuras alternativas dobles anidadas. La sintaxis es la siguiente:

```
if condición1 then
   instrucciones1;
elseif condición2 then
   instrucciones2;
elseif condición3 then
   instrucciones3;
…
[else
  instruccionesn;]
end if;
```



El funcionamiento es el siguiente:

- Se comprueba condición$_1$. Si se cumple, se ejecuta instrucciones$_1$ y termina la sentencia if.
- Si no se cumple condición$_1$, se evalúa condición$_2$. Si se cumple, se ejecuta instrucciones$_2$ y termina la sentencia if.

- Si no se cumple condición$_2$, se evalúa condición$_3$…
- Si no se cumple ninguna de las condiciones especificadas y hay cláusula else, se ejecutan sus instrucciones y finaliza la sentencia if.

Vamos a crear a continuación un procedimiento que actualice el salario del empleado cuyo apellido se pasa como parámetro en función de su oficio. Si es presidente, le subiremos el sueldo un 2 %; si es director, un 3 %; si es vendedor, un 4 %, y si tiene cualquier otro oficio, un 5 %. Para ello, vamos a declarar una variable en la que vamos a almacenar el porcentaje de incremento del salario en función del oficio. En primer lugar, deberemos obtener el oficio del empleado y preguntar por su oficio en una estructura alternativa múltiple. El procedimiento quedará como sigue:

```
mysql> create procedure SubirSalario (ape varchar(40))
    -> begin
    -> declare ofi enum('EMPLEADO', 'VENDEDOR', 'DIRECTOR', 'ANALISTA',
                        'PROGRAMADOR', 'PRESIDENTE');
    -> declare porcen int;
    -> select oficio into ofi from emple where apellido=ape;
    -> if ofi='PRESIDENTE' then
    ->     set porcen = 2;
    -> elseif ofi='DIRECTOR' then
    ->     set porcen = 3;
    -> elseif ofi='VENDEDOR' then
    ->     set porcen = 4;
    -> else
    ->     set porcen = 5;
    -> end if;
    -> update emple set salario=salario+salario*porcen/100
    -> where apellido=ape;
    -> select concat ('Se ha subido el salario a ', ape, ' un ',
    ->               porcen, ' %') Mensaje;
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

Ejecutemos este procedimiento para incrementar el salario del empleado apellidado SALA. Como se trata de un vendedor, su salario se verá incrementado un 4 %.

```
mysql> call SubirSalario ('SALA');//
+--------------------------------------+
| Mensaje                              |
+--------------------------------------+
| Se ha subido el salario a SALA un 4 % |
+--------------------------------------+
1 row in set (0.11 sec)
```

- Estructura alternativa múltiple con CASE de comprobación: esta estructura presenta la siguiente sintaxis:

```
case expresión
when valor1 then
   instrucciones1;
when valor2 then
   instrucciones2;
when valor3 then
   instrucciones3;
…
[else
   instruccionesn+1;]
end case;
```

El funcionamiento es el siguiente:

- Se evalúa expresión.
- Si el valor de expresión coincide con $valor_1$, se ejecuta $instrucciones_1$ y termina la sentencia case; si no,
- Si el valor de expresión coincide con $valor_2$, se ejecuta $instrucciones_2$ y termina la sentencia case; si no,
- Si el valor de expresión coincide con $valor_3$, se ejecuta $instrucciones_3$ y termina la sentencia case; si no…
- Si el valor de expresión no coincide con ninguno de los $valor_i$ y hay cláusula else, se ejecuta $instrucciones_{n+1}$ y finaliza la sentencia case.

Vamos a reescribir el procedimiento del apartado anterior con una sentencia case. Ambos procedimientos son equivalentes.

```
mysql> create procedure SubirSalario2 (ape varchar(40))
    -> begin
    -> declare ofi enum('EMPLEADO', 'VENDEDOR', 'DIRECTOR', 'ANALISTA',
                        'PROGRAMADOR', 'PRESIDENTE');
    -> declare porcen int;
    -> select oficio into ofi from emple where apellido=ape;
    -> case ofi
    -> when 'PRESIDENTE' then
    ->     set porcen = 2;
    -> when 'DIRECTOR' then
    ->     set porcen = 3;
    -> when 'VENDEDOR' then
    ->     set porcen = 4;
    -> else
    ->     set porcen = 5;
    -> end case;
    -> update emple set salario=salario+salario*porcen/100
    -> where apellido=ape;
    -> select concat ('Se ha subido el salario a ', ape, ' un ',
    ->               porcen, ' %') Mensaje;
    -> end;
    -> //
```

- Estructura alternativa múltiple con CASE de búsqueda: esta estructura presenta la siguiente sintaxis:

```
case
when condición1 then
    instrucciones1;
when condición2 then
    instrucciones2;
when condición3 then
    instrucciones3;
…
[else
    instruccionesn+1;]
end case;
```

Esta estructura es totalmente equivalente a la estructura alternativa múltiple con if.

Vamos a reescribir el procedimiento anterior con este tipo de sentencia case. De nuevo, los dos procedimientos son totalmente equivalentes.

```
mysql> create procedure SubirSalario3 (ape varchar(40))
    -> begin
    -> declare ofi enum('EMPLEADO', 'VENDEDOR', 'DIRECTOR', 'ANALISTA',
                        'PROGRAMADOR', 'PRESIDENTE');
    -> declare porcen int;
    -> select oficio into ofi from emple where apellido=ape;
    -> case
    -> when ofi='PRESIDENTE' then
    ->     set porcen = 2;
    -> when ofi='DIRECTOR' then
    ->     set porcen = 3;
    -> when ofi='VENDEDOR' then
    ->     set porcen = 4;
    -> else
    ->     set porcen = 5;
    -> end case;
    -> update emple set salario=salario+salario*porcen/100
    -> where apellido=ape;
    -> select concat ('Se ha subido el salario a ', ape, ' un ',
    ->                porcen, ' %') Mensaje;
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

5.2. Estructura repetitiva

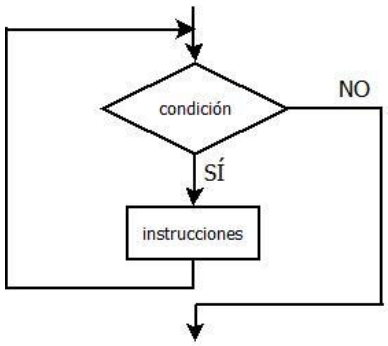
La estructura repetitiva es la otra de las estructuras de control necesarias en todo lenguaje de programación. Una estructura repetitiva permite repetir una secuencia de instrucciones un número determinado de veces. Se pueden emplear con MySQL varias estructuras repetitivas, de las cuales vamos a estudiar dos:

- Estructura repetitiva WHILE: se trata de una estructura repetitiva de 0 a n, lo que quiere decir que el conjunto de instrucciones que forman parte del bucle puede que no se ejecute ninguna vez, una vez o un número indeterminado de veces.

  En la estructura repetitiva while se comprueba una determinada condición. Si se cumple, se ejecuta una secuencia de instrucciones; en caso contrario, finaliza el bucle. Tras ejecutar la secuencia de instrucciones, se vuelve a comprobar la condición y si se cumple, se vuelve a ejecutar la secuencia de instrucciones, y así sucesivamente. En el momento en que no se cumpla la condición, finaliza el bucle.

  La sintaxis de esta estructura es la siguiente:

```
while condición do
   instrucción1;
   instrucción2;
   …
   instrucciónn;
end while;
```



  Vamos a crear a continuación un procedimiento con una sentencia while que reciba un número entero y se encargue de mostrar en pantalla la suma de todos los números enteros entre 1 y el pasado como parámetro (ambos incluidos). Este procedimiento, lo primero que hará es comprobar si el número introducido es cero o negativo, porque en tal caso, se deberá mostrar un mensaje de error. En caso contrario, se deberá realizar la suma de todos los números entre 1 y *maximo* (parámetro recibido).

  Para calcular esta suma necesitamos dos variables:

  - Una, a la que se ha llamado *i*, la cual va conteniendo todos los números comprendidos entre 1 y *maximo.* Por este motivo, esta variable se inicializa con valor 1 y en cada pasada por el bucle se incrementa en una unidad y así sucesivamente mientras que *i* sea menor o igual que *maximo* porque cuando *i* tome un valor mayor que *maximo* no deberemos seguir realizando sumas.
  - Otra variable, a la que se ha llamado *suma*, la cual va a ir conteniendo la suma de los números en cada momento. Esta variable habrá de inicializarse con valor 0 y en cada pasada por el bucle se incrementará con el valor de *i*. De esta manera, al salir del bucle tomará como valor la suma de los números entre 1 y *máximo*. Por ello, al salirse del bucle se muestra en pantalla su valor.

  Se muestra a continuación el código del procedimiento y dos llamadas al mismo:

- En la primera de ellas se llama al procedimiento pasándole como parámetro un número negativo (-3), motivo por el cual debe mostrar un mensaje de error en pantalla.
- En la segunda se llama al procedimiento con valor 6, por lo que el procedimiento debe calcular la siguiente suma: 1+2+3+4+5+6, que es 21, como efectivamente se muestra en pantalla.

```
mysql> create procedure SumarNumeros (maximo int)
    -> begin
    -> declare i int default 1;
    -> declare suma int default 0;
    -> if maximo <= 0 then
    ->     select 'El número pasado como parámetro debe ser mayor que 0'
                  Mensaje;
    -> else
    ->      while i <= maximo do
    ->          set suma = suma + i;
    ->          set i = i + 1;
    ->      end while;
    ->      select concat('La suma de los números entre 1 y ',maximo,' es ',suma) Mensaje;
    -> end if;
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> call SumarNumeros (-3);//
+-----------------------------------------------------+
| Mensaje                                             |
+-----------------------------------------------------+
| El número pasado como parámetro debe ser mayor que 0 |
+-----------------------------------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.04 sec)

mysql> call SumarNumeros (6);//
 +-----------------------------------------+
 | Mensaje                                 |
 +-----------------------------------------+
 | La suma de los números entre 1 y 6 es 21  |
 +-----------------------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.03 sec)
```

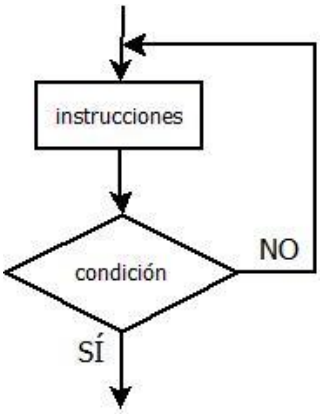
- Estructura repetitiva REPEAT: se trata de una estructura repetitiva de 1 a n, lo que quiere decir que el conjunto de instrucciones que forman parte del bucle como mínimo se va a ejecutar una vez.

  En la estructura repetitiva repeat, en primer lugar, se ejecuta el conjunto de instrucciones que forman parte del bucle y a continuación se comprueba la condición especificada después de la palabra *until*. Si se cumple esta condición, finaliza el bucle; en caso

contrario, se vuelven a ejecutar las instrucciones que forman parte del bucle, y así sucesivamente.

La sintaxis de esta estructura es la siguiente:

```
repeat
   instrucción1;
   instrucción2;
   …
   instrucciónn;
until condicion
end repeat;
```



Vamos a crear una versión del procedimiento *SumarNumeros* con una estructura repetitiva *repeat* en lugar de con *while*. En caso de que al procedimiento se le pase un número inferior o igual a cero, mostraremos un mensaje de error; en caso contrario, ejecutaremos el bucle, en el que deberemos realizar lo mismo que en la anterior versión. En este caso, la condición del *until* debe ser la que determine la salida del bucle, que es que *i* tome un valor mayor que *maximo*. Se muestran a continuación el código del procedimiento y dos ejemplos de llamada al mismo:

```
mysql> create procedure SumarNumeros2 (maximo int)
    -> begin
    -> declare i int default 1;
    -> declare suma int default 0;
    -> if maximo <= 0 then
    ->       select 'El número pasado como parámetro debe ser mayor o
                    igual que 1' Mensaje;
    -> else
    ->     repeat
    ->         set suma = suma + i;
    ->         set i = i + 1;
    ->     until i > maximo
    ->     end repeat;
     ->     select concat('La suma de los números entre 1 y ',maximo,' es ',suma) Mensaje;
    -> end if;
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> call SumarNumeros2(0);//
+-------------------------------------------------------------+
| Mensaje                                                     |
+-------------------------------------------------------------+
| El número pasado como parámetro debe ser mayor o igual que 1 |
+-------------------------------------------------------------+
1 row in set (0.06 sec)

Query OK, 0 rows affected (0.10 sec)
```

```
mysql> call SumarNumeros2(6);//
 +-----------------------------------------+
 | Mensaje                                 |
 +-----------------------------------------+
 | La suma de los números entre 1 y 6 es 21  |
+-----------------------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.03 sec)
```

6. Parámetros.

El paso de información a los subprogramas se realiza por medio de los parámetros. Hemos de distinguir entre parámetros formales y parámetros reales o actuales:

- Los parámetros formales son los que aparecen declarados en la cabecera del procedimiento o función. Por cada uno de ellos recordemos que se debe especificar el nombre y el tipo de dato asociado.
- Los parámetros actuales o reales son los que aparecen en la llamada al procedimiento o función.

Los tipos de los parámetros formales y actuales deben ser compatibles.

El paso de parámetros a los procedimientos y funciones de MySQL se realiza empleando la notación posicional, por la cual cada parámetro actual se asocia con cada parámetro formal según las posiciones que estos ocupan, es decir, el primer parámetro formal tomará el valor del primer parámetro actual, el segundo parámetro formal tomará el valor del segundo parámetro actual, y así sucesivamente.

Los parámetros que se pasan a los procedimientos y funciones pueden ser de entrada, de salida o bien de entrada/salida. Veamos el significado de los diferentes tipos de parámetros:

- Los parámetros de entrada (IN) hacen referencia a los datos que se pasan a un subprograma para que este efectúe operaciones con ellos. En el subprograma llamado no se le puede asignar ningún valor al parámetro formal, sino que solamente se puede utilizar el valor que tiene dicho parámetro. El parámetro actual o real puede ser una variable, una constante o una expresión, cuyo valor se asigna al correspondiente parámetro formal.
- Los parámetros de salida (OUT) se usan para devolver datos del subprograma llamado al que realizó la llamada. El parámetro actual debe ser obligatoriamente una variable a la que se asignará valor en el subprograma llamado para devolverlo al programa llamante.

- Los parámetros de entrada/salida (IN OUT) sirven para pasar un dato del programa llamante al subprograma llamado, modificar dicho valor en el subprograma y devolver el valor modificado al programa llamante. El parámetro actual, como en el anterior caso, debe ser obligatoriamente una variable.

El tipo de parámetro se especifica delante del nombre del mismo. De esta manera una lista de parámetros tendrá la siguiente forma:

```
([tipo1] Nombre_Parámetro1 tipo_dato1, [tipo2] Nombre_Parámetro2 tipo_dato2, …)
```

Los datos $tipo_1$, $tipo_2$,… podrán tomar los siguientes valores:

- IN, para indicar que el parámetro es de entrada.
- OUT, para indicar que el parámetro es de salida.
- INOUT, para indicar que el parámetro es de entrada/salida.

El tipo por defecto es de entrada, por lo que no poner nada equivale a escribir IN como tipo de parámetro. De hecho, todos los parámetros que se han empleado hasta ahora han sido de entrada y por eso no hemos especificado nada antes del nombre de cada parámetro.

A modo de ejemplo, vamos a realizar un procedimiento que reciba la referencia de un pedido y nos devuelva el número de artículos diferentes solicitados en dicho pedido y el número de unidades de artículos solicitadas. La referencia del pedido es un parámetro de entrada y habrá dos parámetros de salida para los otros dos datos indicados. Se crea a continuación el procedimiento:

```
delimiter //
create procedure ConsultarPedido (refPedido char(5), out NArt int,
                                  out NUniArt int)
begin
select count(CodArt), sum(CantArt) into NArt, NUniArt from LineaPedido
where RefPed=refPedido;
select sum(CantArt) into NUniArt from LineaPedido
where RefPed=refPedido;
end;
```

Ahora vamos a hacer otro procedimiento que reciba la referencia de un pedido, llame al procedimiento que acabamos de crear y muestre los datos recibidos:

```
delimiter //
create procedure MostrarInfoPedido(refPedido char(5))
begin
declare numart int;
declare uniart int;
call ConsultarPedido (refPedido, numart, uniart);
select concat ('En el pedido ', refPedido, ' se solicitan ', numart,
       ' artículos distintos, en total ', uniart, ' unidades') Información;
end;
//
```

Si ejecutamos el procedimiento pasando como dato el pedido con código P0001, obtenemos la siguiente salida:

```
mysql> call MostrarInfoPedido('P0001');//
+-------------------------------------------------------------------------------+
| Información                                                                   |
+-------------------------------------------------------------------------------+
| En el pedido P0001 se solicitan 2 artículos distintos, en total 22 unidades |
+-------------------------------------------------------------------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

En este caso, para probar el empleo de parámetros de salida se han creado dos procedimientos, uno de los cuales llama al otro. También podríamos haber probado su funcionamiento mediante lo que se llaman variables de sesión, que son variables a las cuales se les debe asignar un nombre, no se les asigna un tipo y la manera de referirnos a ellas es escribir el nombre de la variable después del símbolo arroba (@). Así, podríamos haber llamado al procedimiento *ConsultarPedido* incluyendo en la llamada dos variables de sesión que tras la ejecución del procedimiento contendrían los valores asignados a dichas variables en la ejecución del procedimiento:

```
mysql> call ConsultarPedido ('P0001', @NumArticulos, @NumUnidades);
Query OK, 1 row affected (0.00 sec)

mysql> select @NumArticulos;
+---------------+
| @NumArticulos |
+---------------+
|             2 |
+---------------+
1 row in set (0.00 sec)

mysql> select @NumUnidades;
+--------------+
| @NumUnidades |
+--------------+
|           22 |
+--------------+
1 row in set (0.00 sec)
```