Apuntes accesibles - Edición de guiones PL/SQL

Sitio: FRANCISCO DE GOYA
Curso: Bases de Datos Pendientes
Libro: Apuntes accesibles - Edición de guiones PL/SQL

Imprimido por: Lucía Hernández Montero
Día: lunes, 24 de octubre de 2022, 09:45

Tabla de contenidos

1. Programas almacenados

En MySql se pueden crear cuatro tipos de programas, siempre dentro de una base de datos:

- Procedimientos: reciben parámetros y funcionan con call.
- Funciones: como los anteriores pero devuelven un valor, por lo que pueden estar en un select.
- Triggers: procedimientos que se desencadenan automáticamente si se da una condición.
- Eventos: similares a los triggers, pero con planificación horaria.

Contienen múltiples líneas (cada una acabada por ”;”), por lo que será necesario cambiar el delimitador.

```
delimiter //
```

Para activar algunos de estos programas, como el trigger y los eventos, será necesario activar el planificador:

```
SET GLOBAL event_scheduler = ON;
```

Puede verse qué procedimientos hay con

```
show procedure status where db = ”miBD”;
```

2. Variables

Se pueden utilizar variables tanto en la consola de mysql como dentro de los programas almacenados.

Las variables globales tienen como prefijo el símbolo "@". Puede accederse y modificarse su valor de las siguientes forma:

```
set @var1 = 5;

select @var1;
```

Las variables locales no necesitan @ previo al nombre, pero han de ser declaradas antes de ser usadas, momento en el que también pueden recibir un valor inicial

```
declare var2 int default 5;

declare leng int;

declare var3 varchar(10);
```

Posteriormente pueden ser usadas en cualquier elemento del código, como un IF, un WHILE, o dentro de un select como condición o tomando valor del mismo o un insert

```
select nombre from Usuario where id = var2 into var3;

select length(var3) into leng;

if leng < 10 then

...
```

3. Procedimientos

Los procedimientos son programas almacenados, que actúan sobre una base de datos.
Reciben parámetros que se declaran como los campos de una tabla. Los parámetros pueden ser IN, OUT o INOUT.

```
create procedure proc1(IN par1 int, OUT par2 varchar(20), INOUT par3 date, par4 int) ...
```

En este ejemplo anterior los parámetros par1 y par4 son de tipo IN, ya que de no indicarse se considera que es de tipo IN.

Hacen acciones entre begin y end.
En el ejemplo, el procedimiento recibe un parámetro ”n int”, el cual funciona como una variable, pudiendo usarse dentro repetidas veces.

```
create table nums(id int auto_increment primary key);
delimiter //
create procedure ins(n int)
begin
insert into nums values(n);
End //
delimiter ;
```

En este caso se usa en un insert.
Si hacemos ”call ins(5)” introducirá el 5 en la tabla nums.

4. Funciones

Las funciones deben devolver un valor, pudiendo recibir parámetros como un procedimiento.
Puede ser de varios tipos, entre otros:

- deterministic: si siempre devuelve el mismo valor usando los mismos parámetros.
- reads sql data: depende de valores en las tablas que use.

En todo caso ha de devolver un valor.

La siguiente función devuelve siempre el mayor valor de la tabla nums:

```
delimiter //

create function mx()
returns int
reads sql data
begin
declare r int default 0;
select max(id) from nums into r;
return r;
end //

delimiter ;
```

En este ejemplo se indica con ”returns” que la función devuelve un int.

Se usa una variable int llamada r. Se le da valor tras un select into. Finalmente se devuelve r.

5. Control de flujo de ejecución

Se muestra en los siguientes epígrafes cómo implementar los elementos comunes de control de flujo de ejecución de cualquier lenguaje, mas uno particular, el cursor.

5.1. IF

En cualquier procedimiento, función, trigger o evento se puede usar el selector if. Un ejemplo sería el siguiente:

```
if a < 4 then
set b = 5;
elseif a < 2 then
set b = 6;
else
set b = 7;
end if;
```

5.2. Repetición

Para repetir acciones mientras se de una condición tendríamos el while:

```
while v1 > 0 do
set a = 1 + a;
set v1 = v1 -1;
end while;
```

Hay otros elementos como loop y repeat:

```
bucle1: LOOP

SET p1 = p1 + 1;

IF p1 >= 10 THEN

 leave bucle1;

END IF;

END LOOP bucle1;
```

El bucle loop solo sale si se llega a la instrucción "leave" con el nombre del bucle. Por otro lado, comienza una nueva iteración del bucle si llega a la instrucción "iterate" con el nombre del bucle.

5.3. Cursores

Los cursores son una herramienta donde se almacena el resultado de una consulta.

Mediante bucles se puede recorrer dicho resultado para así realizar operaciones con cada fila devuelta.

```
fetch cursor into variable;
```

El cursor necesitará un handler, esto es un "manejador", para detectar que ya no hay más filas en el cursor. Habitualmente el handler cambiará el valor de una variable de salida de un bucle.

```
declare done boolean default false;

declare c1 cursor for select a,b from tab1;

declare continue handler for SQLSTATE '02000' set done = TRUE;
```

Ejemplo de cursor

El siguiente cursor se usará en un procedimiento que actuará sobre la
tabla tab1, de arriba a la derecha.
Su funcionamiento es sumar a y b e insertarlo en otra tabla. Pero acumula cada resultado con el siguiente:
4+7 + 0= 11.
1+8 + 11 = 20.
-11+9 +20 = 18.
El resultado está en la tabla tab2.

```
select * from tab1;
```

a	b
4	7
1	8
-11	9

```
select * from tab2;
```

c
11
20
18

Este sería el código completo del cursor.

```
delimiter //
drop procedure if exists procursor//
create procedure procursor()
begin
        declare acumulador int default 0;
        declare x int;
        declare y int;

        declare done boolean default false;
        declare c1 cursor for select a,b from tab1;
        declare continue handler for SQLSTATE '02000' set done = TRUE;

        open c1;
        c1_loop: LOOP
                fetch c1 into x,y;
                if done then
                        leave c1_loop;
                end if;
                set acumulador = x + y + acumulador;
                insert into tab2 values (acumulador);
        end LOOP c1_loop;
        close c1;
end //
delimiter ;
```

6. Triggers

Los triggers son procedimientos que se ejecutan:

- Cuando se da una condición como pueden ser: inserciones, actualizaciones o borrados en una tabla.
- Antes o después de la acción que la desencadenó.
- Si son varias las condiciones que se dan, como puede ser varias inserciones, se ejecutará una vez para cada una de las filas insertadas.

Durante la ejecución se podrá usar:

- old: es una variable que almacena una fila antes de que se elimine o se modifique.
  - old.nombre_col: será la columna de la fila.
- new: es otra variable que almacena la fila que va a ser insertada o a la fila que está a punto de ser actualizada.

En update se podrá usar old o new según sea el trigger. No hay old en un insert, ni new en un delete.

Es importante haber activado el planificador de mysql, cambiando la variable global event_scheduller a ON.

Ejemplo trigger

El siguiente trigger utiliza dos tablas, que se definen a continuación:

```
create table triggers(
   num int primary key,
   let varchar(20));

create table copytrig(
   num int primary key,
   let varchar(20));
```

El trigger tiene una función muy sencilla, y es que cuando se inserte en triggers, se insertará lo mismo (sumando 1 y añadiendo "-R" en copytrig:

```
  delimiter //
drop trigger if exists introtrig //
create trigger introtrig
after insert on triggers for each row
begin
       insert into copytrig
       values(new.num+1,concat("R-",new.let));
end //
delimiter ;
```

El resultado será el siguiente, tras insertar:

```
insert into triggers values (1,"A");

select * from triggers;
```

num	let
1	A

En la otra tabla tendremos:

```
select * from copytrig;
```

num	let
2	R-A

Segundo ejemplo trigger

```
delimiter //
drop trigger if exists updatrig //
create trigger updatrig
before update on triggers for each row
begin
        declare num int default 0;
        select count(*) from triggers where new.let = let into num;
        if (num > 0) then
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Simulando un UNIQUE';
        end if;
end //
delimiter ;
```

Realizar un update, el trigger impedirá que se lleve a cabo, simulando un unique

```
select * from triggers;
```

num	let
1	A
2	B

```
update triggers set let = "B" where num = 1;
```

```
ERROR 1664 (45000): Simulando un UNIQUE
```

En el ejemplo anterior, al ejecutarse antes de actualizar, se puede detener la inserción o cambiarla. El código que lo indica es "before update on triggers"

Si fuese after update, no podría hacerse esa acción de impedir la actualización.

7. Eventos

Los eventos son procedimientos que se ejecutan automáticamente en momentos programados, no se les llama.

Un evento se crea asociado a una BD y a un usuario para los permisos.

La cláusula ON SCHEDULE permite establecer cómo y cuándo se ejecutará el evento. Una sola vez, durante un intervalo, cada cierto tiempo o en una fecha hora de inicio y fin determinadas.

DEFINER especifica el usuario cuyos permisos se tendrán en cuenta en la ejecución del evento.

Ejemplo:

```
drop event creapar;
delimiter //
create event
creapar on schedule every 5 second
starts "2020-01-30 11:25:00"
enable
do
begin
        declare p int;
        select max(num) from par into p;
        insert into par values (p+2);
end //
delimiter ;
```

La variable global event_scheduler por defecto está en off. Para que funcionen hay que cambiarla a on:

```
set global event_scheduler=on;
```

Con disable o con enable se consigue que el evento esté activado o desactivado cuando se cree (sin modificar event_scheduler).

Para activar o desactivar:

```
alter event creapar disable;

alter event creapar enable;
```

Otro ejemplo

```
drop event creaimpar;
delimiter //
create event creaimpar on schedule at
current_timestamp + interval 1 second
enable
do
begin
        declare p int;
        select max(num) from impar into p;
        insert into impar values (p+2);
end //
delimiter ;
```