

Procedimientos almacenados

Los procedimientos almacenados son rutinas invocadas mediante una sentencia CALL. Puede tener parámetros de entrada, de salida o ambos.

La sintaxis simplificada para **crear un procedimiento almacenado** es:

```
CREATE [OR REPLACE] [DEFINER = {user | CURRENT_USER | role |  
CURRENT_ROLE}] PROCEDURE [IF NOT EXISTS] nombre_procedimiento  
([parámetro [,...]]) definición_procedimiento;
```

1

- Siendo:
 - **parámetro:** [IN | OUT | INOUT] nombre_parametro tipo_parámetro
 - **nombre_procedimiento:** el nombre asignado al procedimiento.
 - **definición_procedimiento:** una sentencia SQL válida.
- Ventajas de usar procedimientos almacenados:
 - Ofrecen mejor rendimiento, pues están precompilados.
 - Se ejecutan directamente en el servidor de la base de datos, por lo que se reduce el tráfico a través de la red.
 - Puede usarse para limitar a ciertos usuarios el acceso a determinadas tablas.
- Ejemplos:

```
DELIMITER //  
CREATE PROCEDURE mostrarEmp ()  
BEGIN  
    SELECT * FROM emp;  
END;  
//  
DELIMITER ;
```

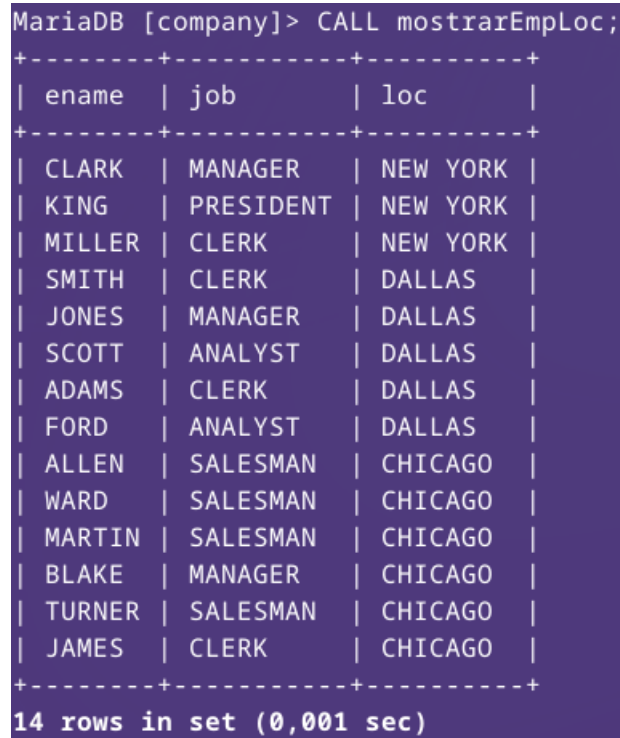
```
DELIMITER //  
CREATE PROCEDURE mostrarEmpLoc ()  
BEGIN  
    SELECT ename, job, loc FROM emp NATURAL JOIN dept;  
END;  
//  
DELIMITER ;
```

Invocar un procedimiento almacenado

Para invocar un procedimiento almacenado se usa la sentencia **CALL**:

CALL nombre_procedimiento **[[argumentos]]**;

- Ejemplo:



MariaDB [company]> CALL mostrarEmpLoc;

ename	job	loc
CLARK	MANAGER	NEW YORK
KING	PRESIDENT	NEW YORK
MILLER	CLERK	NEW YORK
SMITH	CLERK	DALLAS
JONES	MANAGER	DALLAS
SCOTT	ANALYST	DALLAS
ADAMS	CLERK	DALLAS
FORD	ANALYST	DALLAS
ALLEN	SALESMAN	CHICAGO
WARD	SALESMAN	CHICAGO
MARTIN	SALESMAN	CHICAGO
BLAKE	MANAGER	CHICAGO
TURNER	SALESMAN	CHICAGO
JAMES	CLERK	CHICAGO

14 rows in set (0,001 sec)

2

Eliminar un procedimiento almacenado

Para eliminar un procedimiento almacenado se usa la sentencia **DROP PROCEDURE**:

DROP PROCEDURE **[IF EXISTS]** nombre_procedimiento

- Ejemplo:

```
DROP PROCEDURE mostrarEmp;
```

Modificar un procedimiento almacenado

Si bien existe la sentencia **ALTER PROCEDURE**, no nos permite modificar las sentencias que contiene el procedimiento almacenado.

Desde la versión 10.1.3 de MariaDB se introduce el modificador opcional **OR REPLACE** en la sentencia **CREATE**, de forma que, si queremos redefinir un

procedimiento almacenado podemos optar por realizar una sentencia de creación con el mismo nombre que el procedimiento que queremos reemplazar.

- Ejemplo:

```
DELIMITER //
CREATE OR REPLACE PROCEDURE mostrarEmpLoc ()
BEGIN
    SELECT ename, job, hiredate, loc FROM emp NATURAL JOIN dept;
END;
//
DELIMITER ;
```

3

Consultar los procedimientos almacenados

SHOW CREATE PROCEDURE nombre_procedimiento;

- Ejemplo:

```
SHOW CREATE PROCEDURE mostrarEmpLoc;
```

```
MariaDB [company]> SHOW CREATE PROCEDURE mostrarEmpLoc;
+-----+-----+-----+-----+
| Procedure | sql_mode |
+-----+-----+-----+-----+
| Create Procedure | character_set_client | collation_connection | Database Collation |
+-----+-----+-----+-----+
| mostrarEmpLoc | STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+-----+-----+-----+-----+
CREATE DEFINER='root'@'localhost' PROCEDURE `mostrarEmpLoc`()
BEGIN
    SELECT ename, job, loc FROM emp NATURAL JOIN dept;
END | utf8mb3 | utf8mb3_general_ci | utf8mb4_general_ci |
+-----+-----+-----+-----+
1 row in set (0,000 sec)
```

SHOW PROCEDURE STATUS;

- Muestra todos los procedimientos almacenados, incluso los del sistema.

SHOW PROCEDURE STATUS LIKE 'patrón';

- Muestra los procedimientos almacenados cuyo nombre coincida con el patrón proporcionado.
- Ejemplo:

```
SHOW PROCEDURE STATUS LIKE '%emp%';
```

```
MariaDB [company]> SHOW PROCEDURE STATUS LIKE '%emp%';
```

Db	Name	Type	Definer	Modified	Created
company	mostrarEmpLoc	PROCEDURE	root@localhost	2024-03-05 12:00:17	2024-03-05 12:00:17

1 row in set (0,001 sec)



Triggers

Un *trigger*, o disparador, es una sentencia o conjunto de sentencias que se ejecutan (“disparan”) cuando se produce un cierto evento.

Tienen la misma estructura que los procedimientos almacenados, pero a diferencia de éstos, los invoca el sistema ante ciertos eventos, no el usuario.

Creación de *triggers*

```
CREATE [OR REPLACE]
  [DEFINER = {usuario | CURRENT_USER | rol | CURRENT_ROLE}]
  TRIGGER [IF NOT EXISTS] nombre_trigger momento evento
  ON tbl_name FOR EACH ROW
  [{FOLLOWS | PRECEDES} nombre_otro_trigger]
  sentencia_o_sentencias_del_trigger;
```

- Siendo:
 - **nombre_trigger**: nombre del *trigger* que estamos creando.
 - **momento**: el momento en el que se ejecuta el *trigger*. Puede ser:
 - BEFORE: antes del evento.
 - AFTER: después del evento.
 - **evento**: el evento ante el que se dispara el *trigger*. Puede ser:
 - INSERT
 - UPDATE
 - DELETE
- A partir de la versión 10.2.3 de MariaDB se permite asociar más de un *trigger* a un mismo momento de una misma tabla.
- Al programar un *trigger* para las filas de un comando INSERT, UPDATE o DELETE se ejecuta para cada fila. Para controlar el contenido que se pretende modificar se usan dos registros virtuales llamados **OLD** y **NEW**.

- Ejemplo: Vamos a crear una tabla llamada numEmp que almacene el número de empleados que tiene la empresa cada vez que se hace una inserción;

```
CREATE TABLE numEmp (id smallint UNSIGNED AUTO_INCREMENT PRIMARY KEY, numero int UNSIGNED NOT NULL);
```

```
DELIMITER //
```

```
CREATE OR REPLACE TRIGGER actualizarRecuento AFTER INSERT ON emp FOR EACH ROW
```

```
BEGIN
```

```
INSERT INTO numEmp (numero) SELECT COUNT(*) FROM emp;
```

```
END;//
```

```
DELIMITER ;
```

5

```
MariaDB [company]> CREATE TABLE numEmp (id smallint UNSIGNED AUTO_INCREMENT PRIMARY KEY, numero int UNSIGNED NOT NULL);
Query OK, 0 rows affected (0,270 sec)
```

```
MariaDB [company]> describe numEmp;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id    | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| numero | int(10) unsigned    | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0,002 sec)
```

```
MariaDB [company]> SHOW TRIGGERS;
```

```
+-----+-----+-----+-----+-----+-----+
| Trigger          | Event | Table | Statement                                     | Timing |
| Created          |      | sql_mode | Definer          | character_set_client | collation_connection | Database Collation |
+-----+-----+-----+-----+-----+-----+
| actualizarRecuento | INSERT | emp    | BEGIN
  INSERT INTO numEmp (numero) SELECT COUNT(*) FROM emp;
END | AFTER | 2024-03-05 15:55:42.81 | STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE
_SUBSTITUTION | root@localhost | utf8mb3          | utf8mb3_general_ci | utf8mb4_general_ci |
+-----+-----+-----+-----+-----+-----+
1 row in set (0,003 sec)
```

```
MariaDB [company]> INSERT INTO emp (empno,ename,job,mgr,hiredate,sal,comm,deptno) VALUES ('7777','MORGAN','SALES',7902,'2024-02-24',1000,0,0,20);
Query OK, 1 row affected (0,007 sec)
```

```
MariaDB [company]> ;SELECT * FROM numEmp;
ERROR: No query specified
```

```
+-----+-----+
| id | numero |
+-----+-----+
| 1  | 15     |
+-----+-----+
1 row in set (0,000 sec)
```

```
MariaDB [company]> INSERT INTO emp (empno,ename,job,mgr,hiredate,sal,comm,deptno) VALUES ('8888','CROSS','SALES',7777,'2024-02-24',1000.00,0.30);
Query OK, 1 row affected (0,220 sec)

MariaDB [company]> ;SELECT * FROM numEmp;
ERROR: No query specified

+-----+-----+
| id | numero |
+-----+-----+
| 1 | 15 |
| 2 | 16 |
+-----+-----+
2 rows in set (0,000 sec)
```

6

Consultar los *triggers* creados

SHOW TRIGGERS;

SHOW CREATE TRIGGER nombreTrigger;

Borrar un *trigger*

DROP TRIGGER [IF EXISTS] nombreTrigger

Si borramos una tabla también se borrarán los *triggers* que ésta tiene asociados.

Estructuras condicionales

Dentro de un procedimiento almacenado o un *trigger* es útil disponer de estructuras para realizar acciones en respuesta a determinadas condiciones.

En MariaDB podemos usar la estructura IF-THEN-ELSE o CASE:

IF-THEN_ELSE

```
IF condición THEN sentencias
  [ELSEIF otra_condición THEN sentencias] ...
  [ELSE sentencias]
END IF;
```

CASE

```
CASE variable
  WHEN valor THEN sentencias
  [WHEN otro_valor THEN sentencias] ...
  [ELSE sentencias]
END CASE;
```

Estructuras repetitivas

Permiten iterar mientras se cumpla una condición.

WHILE

```
[etiqueta_comienzo:] WHILE condición DO  
    sentencias  
END WHILE [etiqueta_fin]
```



- Se repite la iteración mientras se cumpla la condición
- **etiqueta_fin** debe ser igual a **etiqueta_comienzo**, en caso de que estén presentes.

LOOP

```
[etiqueta_comienzo:] LOOP  
    sentencias  
END LOOP [etiqueta_fin]
```

- Este bucle se repite hasta que dentro se le indique lo contrario, lo cual se realiza mediante una sentencia **LEAVE**.
- Para usar la sentencia LEAVE, hay que asignarle una etiqueta al bucle LOOP, y usar la etiqueta en la sentencia LEAVE. Por ejemplo, un LOOP con etiqueta miBucle detendría su ejecución con la sentencia LEAVE miBucle;

REPEAT

```
[etiqueta_comienzo:] REPEAT  
    sentencias  
UNTIL condición  
END REPEAT [etiqueta_fin]
```

- En este caso el bucle se repite hasta que la condición indicada en UNTIL sea cierta.