PL/MySQL

Introducción

- Extensión procedimental de un lenguaje SQL: es un lenguaje que permite diseñar programas para ser ejecutados dentro de la bbdd y que incluyen instrucciones SQL.
- Todos los SGBD suelen incorporar su propio PL. No existe un estándar, depende del SG elegido. Así, cada uno tiene su propia extensión:
  - Oracle: PL/SQL
  - SQL-Server: Transact-SQL.
  - MySql: PL/MySql.
  - PostgreSQL: PL/PgSQL.

PL/MySql

- Programas almacenados: objetos asociados a una bbdd (esquema) definidos utilizando instrucciones SQL que son alojados en el servidor para una posterior ejecución.
- Tipos:
  - **Rutinas**: incluye procedimientos y funciones. Los procedimientos no devuelven ningún valor. Se utilizan para modificar parámetros que después serán inspeccionados. Las funciones sí devuelven un valor. Se crean para ser invocadas desde cualquier expresión -una consulta, por ejemplo- y que devuelvan un valor que será utilizado en la evaluación de la misma.
  - **Triggers o disparadores**: objeto asociado a una tabla que se 'dispara' cuando ocurre algún evento determinado en la tabla (una modificación, una inserción, etc.)
  - **Events**: tareas que el servidor realiza de forma periódica.

Procedimientos

- Definición: conjunto de instrucciones SQL almacenadas en el servidor.
- Ventajas: el aumento de la velocidad de respuesta del servidor (todo se ejecuta en el servidor) y permitir la creación de librerías de procedimientos (ganando en diseño y modularidad).
- Usos:
  - Cuando diferentes aplicaciones escritas en diferentes plataformas deben realizar las mismas tareas.
  - Cuando la seguridad es una prioridad se puede obligar a los clientes a únicamente utilizar procedimientos albergados. Con ello, se consigue un mayor control de accesos, cambios, etc.

Sintaxis

NOTA: antes de crear el procedimiento debemos tener una bbdd en uso. Los procedimientos están asociados a una bbdd.

CREATE [DEFINER = { user | CURRENT_USER }] PROCEDURE sp_name ([**proc_parameter**[,...]]) [**characteristic** ...] **routine_body**

proc_parameter: [IN | OUT | INOUT] param_name **type**

type: Any valid MySQL data typea

characteristic: COMMENT 'string' | LANGUAGE SQL | [NOT] DETERMINISTIC | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }| SQL SECURITY { DEFINER | INVOKER }

routine_body: Valid SQL routine statement

Problemas: PL1_1

- Trata de escribir un procedimiento. Con que contenga una sola consulta es suficiente.
- Ahora escribe un procedimiento con dos subconsultas.

Invocar procedimientos

```
CALL sp_name([parameter[,...]])
```

O también:

```
CALL sp_name[()]
```

- Siempre debe haber una lista de parámetros. Se puede utilizar una lista vacia
- Los parámetros pueden ser IN, OUT o INOUT. Por defecto son IN. Veamos:
  - Tipo IN: parámetro que se pasa al procedimiento. El procedimiento podrá modificar el valor del parámetro pero dicha modificación no será visible al salir del procedimiento.
  - Tipo OUT: parámetro que si es visible después de haber sido modificado por el procedimiento.
  - Tipo INOUT: parámetro que se pasa (inicializado) al procedimiento y cuyo valor sí será visible después de ejecutarse el procedimiento.

Problemas: PL1_2

- Invoca a los procedimientos anteriores
- Ahora escribe un procedimiento que devuelva el valor numérico que le pasamos como parámetro.

Características

- COMMENT: para añadir comentarios
- LENGUAGE: en teoría para indicar el lenguaje en que está escrito el procedimiento. No tiene validez. MySql sólo acepta SQL.
- [NOT] DETERMINISTIC: si para el mismo conjunto de valores de los parámetros la rutina siempre se comporta de la misma manera, diremos que es una rutina "determinista" si no (por ejemplo, utiliza funciones como RAND() o NOW()), pues "no determinista". Tiene consecuencias a nivel de optimización y replicación que a nosotros no nos afectan (de momento).
- { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }: aporta información sobre la naturaleza de los datos usados por la rutina. No es vinculante. Las opciones (respectivamente) son: no contiene instrucciones de lectura (SELECT) o de escritura (UPDATE); no contiene instrucciones SQL; sólo utiliza instrucciones de lectura; puede contener instrucciones de escritura. Por defecto, la primera.

- SQL SECURITY: para indicar si la rutina se debe ejecutar con los permisos del usuario (DEFINER) indicado por el creador de la rutina o con los permisos del usuario que la invoca (INVOKER). Por defecto es igual a DEFINER. Luego volveremos sobre esto.

→ **routine_body**: Consiste en una instrucción SQL. Ésta instrucción puede ser un simple SELECT o INSERT o una composición de instrucciones SQL escrita utilizando BEGIN...END.

Existen limitaciones: no todas las instrucciones se pueden utilizar dentro de un procedimiento. Por ejemplo, la instrucción USE provoca un error. Ya volveremos sobre esto más adelante.

Problemas: PL1_3

- ¿Qué apartado dedica el manual a los procedimientos?
- ¿Qué permisos son necesarios para ejecutar un procedimiento?
- ¿Y para crearlo?
- Trata de averiguar para qué sirve la característica SQL SECURITY. Haz pruebas y elabora una teoría al respecto.

Privilegios

- CREATE ROUTINE: necesario para crear procedimientos o funciones.
- ALTER ROUTINE: necesario para modificar o eliminar rutinas. Este privilegio es otorgado de forma automática al creador de la rutina y se le revoca cuando la rutina es eliminada.
- EXECUTE: necesario para ejecutar rutinas alojadas en el servidor. Este privilegio es otorgado de forma automática al creador de la rutina y se le revoca cuando la rutina es eliminada.
- Si la variable *automatic_sp_privileges* es 0, los permisos de ALTER ROUTINE y EXECUTE no son automáticamente concedidos al creador de las rutinas.

- Todas las rutinas (y vistas) alojadas en el servidor pueden tener un atributo DEFINER que refiere a una identidad del sistema. Si dicho atributo es omitido en la definición de la vista o de la rutina, la identidad por defecto será la del usuario que crea la rutina o vista.
- Además, procedimientos, vistas y funciones disponen de la característica SQL SECURITY que puede ser DEFINER o INVOKER. Veamos las diferencias:
    - **DEFINER**: el procedimiento/función/vista se ejecutará con los permisos del usuario indicado en el campo DEFINER. Pueden ser unos permisos completamente distintos a los del usuario que invoca la rutina. Éste -el usuario que invoca- sólo necesita poder acceder a la bbdd asociada a la rutina y tener permisos de EXECUTE. Una vez la rutina ha comenzado, los permisos del "invocador" son ignorados y sólo se aplican los de la cuenta DEFINER.
    - **INVOKER**: la rutina se ejecuta con permisos del usuario que la invoca.
- MySql sigue las siguientes reglas para controlar qué identidades se pueden colocar en DEFINER:
    - Para especificar un usuario distinto al tuyo en el atributo DEFINER, debes tener el privilegio SUPER.
    - Si no dispones del privilegio de SUPER, el único usuario legal es el tuyo (CURRENT_USER)

Problemas: PL1_4

- Crea un procedimiento que muestre por pantalla el resultado de consultar la tabla mysql.user. Observa que el DEFINER queda definido como root@localhost. Asegúrate de que la característica SQL SECURITY queda como DEFINER
- Ahora concede al usuario María@'%' permisos para ejecutar rutinas.
- Accede al sistema como María y trata de ejecutar el procedimiento anterior.
- Vuelve a logearte como root y modifica el procedimiento anterior: cambia la característica SQL SECURITY a INVOKER
- Vuelve a convertirte en María y trata de nuevo de ejecutar el procedimiento.
- ¿Qué permisos habría de tener María para poder llamar el procedimiento?

Rutinas y Metadatos

Los metadatos asociados a las rutinas se guardan en la tabla INFORMATION_SCHEMA.ROUTINES. Para acceder a la información que contiene dicha tabla podemos realizar una consulta o utilizar: SHOW CREATE PROCEDURE nombre_procedimiento para visualizar la definición de un procedimiento o SHOW PROCEDURE STATUS nombre_procedimiento para informarnos de sus características.

Bloque de instrucciones

- Para definir bloques de instrucciones utilizamos la siguiente sintaxis:

```
[begin_label:] BEGIN

[statement_list]

END [end_label]
```

- Los bloques de instrucciones pueden aparecer tanto en procedimientos, en funciones, en disparadores o en eventos.
- **statement list:** Estará formado por cero, una o más instrucciones acabadas en punto y coma.

- begin_label / end_label: Se pueden etiquetar los bloques con el fin de, posteriormente, poder trasladarse de un punto a otro del procedimiento. Las reglas de etiquetado son:
  - *begin_label* debe ir seguido de dos puntos
  - Puede haber un *begin_label* sin su *end_label* asociado. Si colocamos el *end_label*, debe llamarse exactamente igual que el *begin_label* al que hace referencia.
  - No puede haber un end_label sin su *begin_label*
  - Etiquetas al mismo nivel de anidamiento deben tener diferentes nombres.
  - Pueden ser de hasta 16 caracteres.

Definición de variables

- Para definirlas utilizamos la instrucción DECLARE:

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

- Si no indicamos valor por defecto adoptará NULL.
- Variables deben ser declaradas al principio del bloque de instrucciones.
- Las reglas para escribir el nombre de la variable son las mismas que para cualquier objeto de la bbdd. Es decir, no irá entre comillas simples si sólo utiliza [0-9,a-z,A-Z,$,_]. Si utiliza otros caracteres habrá de ir entre comillas simples.
- No son case_sensitive.

- La visibilidad de una variable local se ciñe al bloque BEGIN...END en el que se ha definido. Puede ser referenciada por bloques internos excepto si en dichos bloques se declaran variables con el mismo nombre.
- Para asignarle un valor a una variable tenemos dos posibilidades:

```
SELECT ... INTO nombre_variable ...
```

```
SET nombre_variable=...
```

Problemas: PL1_5

1. Procedimiento 1: debe contar el número de usuarios distintos del sistema y lo muestre por pantalla.
2. Procedimiento 1.1: debe insertar un cliente en la tabla empresa.CLIENTE. Recibirá dos parámetros internos que serán el nombre y el código de cliente.
3. Procedimiento 1.2: crear usuario 'Pepe' que sin permisos para insertar en la tabla CLIENTE pueda llamar a el procedimiento anterior y realizar la inserción a través de él.
4. Procedimiento 2:
    a. Parámetros de entrada: fecha_var tipo DATE.
    b. Calcule la diferencia en días entre la fecha actual y la fecha_var.
    c. Mostrar por pantalla fecha_var en formato 'día de la semana dd/mm/aaaa'.
    d. Mostrar también la diferencia de fechas en días calculada previamente.

5. Procedimiento3:
    a. Parámetros de entrada var_1 VARCHAR, var_2 VARCHAR
    b. Crea la tabla prueba_tabla_1 (campo1 valor numérico incremental, campo2 varchar (50)). El //charset// debe ser el adecuado.
    c. Concatena var_1 con var_2 y lo inserta en prueba_tabla_1.campo2 en mayúsculas
    d. Finaliza mostrando por pantalla algo como: "La inserción se ha realizado correctamente. El número de registros en la tabla_1 es: <número de registros>"
6. Procedimiento4: Contabiliza el número de veces que es llamado. Cada vez que es ejecutado aparecerá un mensaje del estilo: 'este procedimiento ha sido llamado XX veces'