FUNCIONES Y PROCEDIMIENTOS ALMACENADOS MYSQL

Son nuevas funcionalidades de la versión de MySQL 5.0. Un procedimiento almacenado o una función es un conjunto de comandos SQL que pueden almacenarse en el servidor. Una vez que se hace, los clientes no necesitan relanzar los comandos individuales pero pueden en su lugar referirse al procedimiento almacenado o función con su respectivo nombre.

Los procedimientos almacenados y las funciones se agrupan en un conjunto llamado rutinas (Routines).

Algunas situaciones en que las funciones o procedimientos almacenados pueden ser particularmente útiles:

- Cuando múltiples aplicaciones cliente se escriben en distintos lenguajes o funcionan en distintas plataformas, pero necesitan realizar la misma operación en la base de datos.

- Cuando la seguridad es muy importante. Los bancos, por ejemplo, usan procedimientos almacenados para todas las oparaciones comunes. Esto proporciona un entorno seguro y consistente. En tal entorno, las aplicaciones y los usuarios no obtendrían ningún acceso directo a las tablas de la base de datos, sólo pueden ejecutar algunos procedimientos almacenados.

- Los procedimientos almacenados pueden mejorar el rendimiento ya que se necesita enviar menos información entre el servidor y el cliente. El intercambio que hay es que aumenta la carga del servidor de la base de datos ya que la mayoría del trabajo se realiza en la parte del servidor y no en el cliente.

Para algunos ejemplos vamos a utilizar la siguiente base de datos como ejemplo.

```
CREATE DATABASE rutinas1;
USE rutinas1;

CREATE TABLE puntos (
  id_participante INT NOT NULL,
  genero CHAR(1) NOT NULL,
  competencia_1 INT  NOT NULL,
  competencia_2 INT NOT NULL,
  PRIMARY KEY (id_participante)
);

INSERT INTO puntos VALUES(1,'M', 9, 7);
INSERT INTO puntos VALUES(2,'F', 6, 8);
INSERT INTO puntos VALUES(3,'M', 9, 9);
INSERT INTO puntos VALUES(4,'F', 10, 7);
INSERT INTO puntos VALUES(5,'M', 9, 10);
```

LAS FUNCIONES

Las funciones devuelven un único valor simple: un integer, un string, o algo similar.

El siguiente es un ejemplo de función que toma un parámetro, realiza una operación con una función SQL, y retorna el resultado:

```
DELIMITER //
CREATE FUNCTION holaMundo() RETURNS VARCHAR(20)
RETURN 'HolaMundo';
//
DELIMITER ;

SELECT holamundo();
SELECT holamundo() as Retorno;
```

Función con variables,

```
DELIMITER //
CREATE FUNCTION holaMundo_var() RETURNS VARCHAR(40)
BEGIN
    DECLARE var_salida VARCHAR(40);
    SET var_salida = 'Hola Mundo desde una Variable';
    RETURN var_salida;
END;
//
DELIMITER ;
```

```
SELECT holaMundo_var();
```

Función con parámetros,

```
DELIMITER //
    CREATE FUNCTION saludo (texto CHAR(20))
    RETURNS CHAR(50)
    RETURN CONCAT('Hola, ',texto,'!');
//
DELIMITER ;
```

Ejecutamos la función enviando parámetros,

```
SELECT saludo('Mundo');
SELECT saludo('Pepito Perez');
```

Para eliminar una función se usa DROP FUNCTION nombrefuncion.

```
DROP FUNCTION holaMundo;
```

Para observar la descripción de una función se usa SHOW CREATE FUNCTION nombrefuncion.

```
SHOW CREATE FUNCTION saludo\G
```

Detalles de todos las funciones almacenadas:

```
SHOW FUNCTION STATUS\G
```

Funciones con más de un parámetro,

```
DELIMITER //
CREATE FUNCTION elmayor (num1 INT, num2 INT)
RETURNS INT
    BEGIN
        DECLARE var_retorno INT;
        IF num1 > num2 THEN
            SET var_retorno = num1;
        ELSE
            SET var_retorno = num2;
        END IF;
        RETURN var_retorno;
    END;
//
DELIMITER ;
```

Ejecutamos la función enviando parámetros,

```
SELECT elmayor(33,25);
SELECT elmayor(12,28);
```

Podemos aplicar esta función dentro de una consulta.

```
SELECT un_campo, otro_campo, lafuncion(un_campo, otro_campo)
as NombreCampo FROM tabla;
```

```
SELECT competencia_1, competencia_2, elmayor(competencia_1, competencia_2)
as Mayor_Puntaje FROM puntos;
```

Función con CASO o CASE,

```
DELIMITER //
CREATE FUNCTION prioridad (cliente_prioridad VARCHAR(1)) RETURNS
VARCHAR(20)
BEGIN
    CASE cliente_prioridad
        WHEN 'A' THEN
        RETURN 'Alto';
        WHEN 'M' THEN
        RETURN 'Medio';
        WHEN 'B' THEN
        RETURN 'Bajo';
        ELSE
        RETURN 'Dato no Valido';
    END CASE;
END
//
DELIMITER ;

SELECT prioridad('M');
SELECT prioridad('B');
SELECT prioridad('S');
```

Función con acceso a datos

```
DELIMITER //
CREATE FUNCTION sumas (p_genero CHAR(1))
RETURNS INT
    BEGIN
        DECLARE var_suma INT;
        SELECT SUM(competencia_1)
        INTO var_suma
        FROM puntos
        WHERE genero = p_genero OR p_genero is NULL;
        RETURN var_suma;
    END;
//
DELIMITER ;


SELECT sumas('M');
SELECT 'Hombres' as Genero, sumas('M') as Total_Compe1;
SELECT 'Mujeres' as Genero, sumas('F') as Total_Compe1;
SELECT 'Todos' as Genero, sumas(NULL) as Total_Compe1;
```

Mostramos una sola tabla con todos los datos.

```
SELECT 'Hombres' as Genero, sumas('M') as Total_Compe1
UNION
SELECT 'Mujeres' as Genero, sumas('F') as Total_Compe1
UNION
SELECT 'Todos' as Genero, sumas(NULL) as Total_Compe1;
```

Ejercicio, realizar una función que calcule el total de puntos de las dos competencias para cada uno de los registros y mostrarlos en una nueva consulta.

Funciones con ciclos.

Realizar una función que acepte un número y una potencia, ejecute la operación usando un ciclo y devuelva el resultado.

```
DELIMITER //
CREATE FUNCTION res_potencia(numero INT, potencia INT) RETURNS INT
BEGIN
    DECLARE contador INT DEFAULT 1;
    DECLARE resultado INT DEFAULT 1;
    WHILE contador <= potencia DO
        SET resultado = resultado * numero ;
        SET contador = contador + 1;
    END WHILE;
    RETURN resultado;
END;
//
DELIMITER ;
```

Llamamos la función enviando parámetros.

```
SELECT res_potencia(2,3);
SELECT res_potencia(3,3);
SELECT res_potencia(2,4);
```

Ejercicio, realizar una función que enviado un número, devuelva el resultado de sumar sus 5 primeros múltiplos, usando ciclos.
Ejemplo si envió por parámetros el 7
Resultado=(7*1)+7*2)+(7*3)+(7*4)+(7*5)

Ejercicio, realizar una función que enviado tres notas de 0.0 a 5.0 por parámetros, devuelva el resultado de la nota final, teniendo en cuenta que la nota1 vale el 20% la nota2 vale el 30% la nota3 vale 50]%.

Ejemplo si envió por parámetros notas(4.5,3.0,4.7)
La NotaFinal= 4.2

Luego de Realizar la Función, utilizarla con un INSERT para completar la siguiente Tabla.

```
CREATE TABLE nota (
          id INT(4) PRIMARY KEY AUTO_INCREMENT,
          nota1 FLOAT(2,1) DEFAULT 0,
          nota2 FLOAT(2,1) DEFAULT 0,
          nota3 FLOAT(2,1) DEFAULT 0,
          final FLOAT(2,1) DEFAULT NULL
        );

INSERT INTO nota (nota1, nota2, nota3) VALUES(2.5,3.5,4.0);
INSERT INTO nota (nota1, nota2, nota3) VALUES(4.5,5.0,4.0);
INSERT INTO nota (nota1, nota2, nota3) VALUES(1.5,4.5,3.0);
INSERT INTO nota (nota1, nota2, nota3) VALUES(3.5,3.8,4.4);
INSERT INTO nota (nota1, nota2, nota3) VALUES(3.9,3.7,4.9);

SELECT * FROM nota;
```

Procedimientos

Son también bloques de código que a diferencia de las funciones no devuelven ningún valor. Bajo esta premisa su cometido es ligeramente distinto, puesto que no espera un resultado tras finalizar su ejecución, aunque las acciones que haga el procedimiento y las posibles modificaciones de los datos que realice puedan verse como el resultado de su ejecución.

Sirven para realizar tareas (agregar, modificar o borrar registros), o devolver resultados en forma de tablas.

Ejemplo, consultamos los registros de la tabla puntos con ayuda de un procedimiento almacenado.

Primero creamos el procedimiento.

```
DELIMITER //
CREATE PROCEDURE listar_tabla ()
        SELECT * FROM puntos;
//
DELIMITER ;
```

Luego llamamos el procedimiento con el comando CALL

```
CALL listar_tabla;
```

Procedimiento con parámetros,

```
DELIMITER //
CREATE PROCEDURE consultar_puntos2 (p_id INT)
       SELECT * FROM puntos
       WHERE id_participante = p_id OR p_id is NULL;
//
DELIMITER ;
```

Hacemos la llamada al procedimiento enviamos parámetros.

```
CALL consultar(NULL);
CALL consultar(3);
```

Podemos usar cualquier tipo de consulta.

```
DELIMITER //
CREATE PROCEDURE buscar_puntos(num INT)
    SELECT * FROM puntos
    WHERE competencia_2 > num;
//
DELIMITER ;

CALL buscar_puntos(8);
```

También podemos realizar procedimientos para Insertar datos.

```
DELIMITER //
CREATE PROCEDURE  insertar_puntos(
                                    nuevo_id INT,
                                    nuevo_genero CHAR(1),
                                    nuevo_puntos_1 INT,
                                    nuevo_puntos_2 INT
                                  )
        INSERT INTO puntos (
            id_participante, genero, competencia_1, competencia_2)
        VALUES (
            nuevo_id, nuevo_genero, nuevo_puntos_1, nuevo_puntos_2);
//
DELIMITER ;
```

Luego llamamos el procedimiento y le enviamos parámetros.

```
CALL insertar_puntos(6,'F', 10, 10);

SELECT * FROM puntos;
```

Uso de parámetros IN, OUT

```
DELIMITER //
CREATE PROCEDURE genero_cantidad(IN letra CHAR(1), OUT cantidad INT)
BEGIN
    SELECT * FROM puntos
    WHERE genero = letra;
    SELECT COUNT(*) INTO cantidad
    FROM puntos
    WHERE genero = letra;
END
//
DELIMITER ;

CALL genero_cantidad('M', @cantidad);

SELECT @cantidad;
```

Para eliminar un PROCEDIMIENTO se usa DROP PROCEDURE nombre.

```
DROP PROCEDURE listar_tabla;
```

Para observar la descripción de un PROCEDIMIENTO se usa SHOW CREATE PROCEDURE nombre.

```
SHOW CREATE PROCEDURE genero_cantidad\G
```

Ver Detalles de todos los Procedimientos Almacenados:

```
SHOW PROCEDURE STATUS\G
```

Ejercicio, Con las siguientes tablas.

```
CREATE TABLE articulos (
  id_articulo INT NOT NULL,
  descripcion VARCHAR(30)  NOT NULL,
  precio FLOAT  NULL,
  auditoria DATETIME NOT NULL,
  PRIMARY KEY (id_articulo)
);

INSERT INTO articulos VALUES(1, 'Leche 1L.', 2000, '2013-09-23 12:23:14');
INSERT INTO articulos VALUES(2, 'Café 250 gr.', 2400, '2013-10-22 18:33:51');
INSERT INTO articulos VALUES(3, 'Agua 5L.', 3900, '2013-10-01 20:16:36');
INSERT INTO articulos VALUES(4, 'Galletas 200 gr.', 1000, '2013-08-11
10:03:54');


CREATE TABLE novedades (
  id_articulo INT NOT NULL,
  descripcion VARCHAR(30)  NOT NULL,
  precio FLOAT DEFAULT NULL
);

INSERT INTO novedades VALUES(2, 'Café 250 gr.', 2500);
INSERT INTO novedades VALUES(3, 'Agua 5L.', 4000);
```

Programe un procedimiento que actualice el precio de los productos de la tabla artículos en función de los precios que indica la tabla novedades.