DOCUMENTACION TALLER #5 ACTIVIDAD 3:

Para comenzar a realizar esta actividad, con el fin de poder crear 4 procedimientos almacenados que me permitieran agregar, actualizar, consultar y borrar, lo primero que hice para esto fue seleccionar la tabla de mi interés , la cual fue editorial y entonces tras haber hecho aquella selección , empecé con lo bueno :

Primero que todo, repasé la estructura adecuada para crear los procedimientos:

- Un DELIMITER para saber dónde empieza y donde es que termina
- Luego usé la sentencia CREATE PROCEDURE que es la instrucción para crear un nuevo procedimiento almacenado en la base de datos + (nombre del procedimiento) +(IN o OUT para indicar si los parámetros del procedimiento son de entrada o de salida + 1 o varios datos de entrada o procedimientos + TIPO DE DATO)
- Luego hacer uso del BEGIN y el END, que son comandos que se utilizan para indicar el comienzo y el final del cuerpo del procedimiento
- Y finalmente, como dato informativo que aprendí es que, dentro del cuerpo del procedimiento, se puede escribir cualquier sentencia SQL que sea necesaria, como SELECT, INSERT, UPDATE, DELETE, entre otras. También se pueden usar condicionales, ciclos y declarar variables locales para hacer las operaciones.

Y, luego de repasar estos datos para hacer la actividad bien, empecé con el primer procedimiento:

1. Procedimiento para agregar un registro de editoriales:

Para hacer este y los demás procedimientos, hago los siguientes pasos:

Primero uso el DELIMITER //

Luego creo el procedimiento almacenado llamado "AgregarEditorial". El "IN" que aparece antes de cada parámetro indica que son parámetros de entrada.

Luego escribo mis parámetros de entrada: IN p_nombre VARCHAR (50), IN p_ciudad VARCHAR (30), IN p_complemento VARCHAR(100), IN p_telefono VARCHAR(20). Cada uno tiene un nombre (p_nombre, p_ciudad, p_complemento, p_telefono) y un tipo de datos (VARCHAR) con una longitud máxima específicada anteriormente.

Uso la sentencia BEGIN

Luego indico que es lo que voy a insertar y en donde INSERT INTO Editorial (nombre, ciudad, complemento, Telefono) y luego enumero las columnas que se van a actualizar.

Ya casi finalizando indico los valores que se van a insertar en la tabla "Editorial" para las columnas respectivas: "nombre", "ciudad", "complemento" y "Telefono".VALUES (p_nombre, p_ciudad, p_complemento, p_telefono)

Finalizo el cuerpo del procedimiento almacenado con END

Y finalmente dejo todo listo con DELIMITER para terminar la creación del procedimiento almacenado.

Luego, con la sentencia CALL hago un llamado a mi procedimiento e ingreso los valores:

```
286    -- procedimiento para agregar un resgitro de editoriales:
287    DELIMITER //
288    CREATE PROCEDURE AgregarEditorial(
289      IN p_nombre VARCHAR(50),
290      IN p_ciudad VARCHAR(30),
291      IN p_complemento VARCHAR(100),
292      IN p_telefono VARCHAR(20)
293    )
294    BEGIN
295      INSERT INTO Editorial(nombre, ciudad, complemento, Telefono)
296      VALUES(p_nombre, p_ciudad, p_complemento, p_telefono);
297    END //
298    DELIMITER ;
299
300    CALL AgregarEditorial('Colombina','Buenos Aires','edificio Santorini','5557896-1');
301
```

2. Procedimiento para actualizar un registro de editoriales:

Lo primero que hice fue identificar que tenía cuatro parámetros de entrada, todos ellos de tipo VARCHAR: "p_nombre", "p_ciudad", "p_complemento" y "p_telefono", luego el procedimiento que voy a realizar es una operación UPDATE en la tabla "Editorial" utilizando los valores pasados como parámetros, y finalmente el registro que se actualizare es aquel cuyo campo "nombre" coincide con el valor pasado en el parámetro "p_nombre". Así fue como me quedo la sentencia de mi procedimiento llamándolo con la sentencia CALL y los valores esperados :

```
302    -- procedimiento para actualizar un resgitro de editoriales:
303    DELIMITER //
304    CREATE PROCEDURE ActualizarEditorial(
305      IN p_nombre VARCHAR(50),
306      IN p_ciudad VARCHAR(30),
307      IN p_complemento VARCHAR(100),
308      IN p_telefono VARCHAR(20)
309    )
310    BEGIN
311      UPDATE Editorial
312      SET ciudad = p_ciudad, complemento = p_complemento, Telefono = p_telefono
313      WHERE nombre = p_nombre;
314    END //
315    DELIMITER ;
316
317    CALL ActualizarEditorial('Colombina','Buenos Aires','edificio Santorini','5578960-2');
318
```

3. Procedimiento para consultar un registro:

Creé un código que crea un procedimiento almacenado llamado "ConsultarEditorial" quelo que hace es recibe un parámetro de entrada "p_nombre" de tipo VARCHAR(50), y devuelve todas las filas de la tabla "Editorial" donde la columna "nombre" coincide con el valor del parámetro. Finalmente , invoco mi procedimiento haciendo uso de la sentencia CALL.

```
319     -- procedimiento para consultar un resgitro de editoriales:
320     DELIMITER //
321     CREATE PROCEDURE ConsultarEditorial(
322       IN p_nombre VARCHAR(50)
323     )
324     BEGIN
325       SELECT * FROM Editorial
326       WHERE nombre = p_nombre;
327     END //
328     DELIMITER ;
329
330     DROP procedure ConsultarEditorial;
331     CALL ConsultarEditorial('Colombina');
```

```
1   CALL ConsultarEditorial('Colombina');
2
```

sult Grid | Filter Rows: | Export: | Wrap Cell Content:

nombre	ciudad	complemento	Telefono
Colombina	Buenos Aires	edificio Santorini	5578960-2

4. Procedimiento para borrar un registro:

Lo que hice fue crear el procedimiento almacenado llamado "BorrarEditorial" con un parámetro de entrada llamado "p_nombre" y un tipo de dato "VARCHAR (50)". Esto quiere decirme que cuando llamo a este procedimiento almacenado, debo proporcionar un valor para el parámetro "p_nombre", que se utilizará para identificar el registro que se debe borrar.

Luego, dentro del cuerpo del procedimiento almacenado, la sentencia "DELETE" la utilicé para borrar el registro correspondiente en la tabla "Editorial", y entonces la cláusula "WHERE" la utilicé para especificar qué registro se debía borrar, basado en el valor del parámetro "p_nombre".

Además, luego de crear la estructura de mi procedimiento, lo que hago es llamarlo con la sentencia CALL y los valores necesarios o esperados.

```
333     -- procedimiento para borrar un registro de editoriales :
334     DELIMITER //
335     CREATE PROCEDURE BorrarEditorial(
336       IN p_nombre VARCHAR(50)
337     )
338     BEGIN
339       DELETE FROM Editorial
340       WHERE nombre = p_nombre;
341     END //
342     DELIMITER ;
343     CALL BorrarEditorial('Colombina');
```

Output

Action Output

#	Time	Action	Message
17	20:08:13	CALL ConsultarEditorial('Colombina')	1 row(s) returned
18	20:09:22	CREATE PROCEDURE BorrarEditorial(IN p_nombre VARCHAR(50)) BEGIN DELETE F...	0 row(s) affected
19	20:09:36	CALL BorrarEditorial('Colombina')	1 row(s) affected

Luego de haber creado los procedimientos, lo que procedo a hacer es crear los triggers pero para hacer esto, antes debo informarme muy bien sobre los trigger y luego crear una tabla que me almacenara esos registros ;

Esta fue la información que logre captar sobre su definición: Un trigger en SQL es un tipo de objeto de base de datos que se utiliza para ejecutar automáticamente una serie de instrucciones o procedimientos almacenados en respuesta a un evento específico en una tabla o vista.

Y, los eventos que pueden desencadenar un trigger incluyen cambios en los datos de la tabla (por ejemplo, una inserción, actualización o eliminación de filas), y también ciertos eventos de sistema, como el inicio de sesión de un usuario.

Y para cerciorarme de hacer las cosas bien, identifique unos pasos para la creación de trigger adecuada :

1. Decidir sobre el tipo de evento que activará el trigger, como INSERT, UPDATE o DELETE en una tabla determinada.
2. Crear el procedimiento almacenado o la serie de instrucciones que deseas que se ejecute cuando se active el trigger.
3. Crear el trigger utilizando el comando CREATE TRIGGER, especificando el nombre del trigger, el evento que lo activará y el procedimiento o instrucciones que se deben ejecutar.

esta fue la tabla que creé:

```
345     -- Crear la tabla "control_de_cambios_librería"
346 ●  CREATE TABLE control_de_cambios_librería (
347       usuario VARCHAR(50),
348       accion VARCHAR(10),
349       fecha TIMESTAMP
350   );
351
```

Luego, procedo entonces a crear los triggers:

El primer trigger que creé "insertar_registro" se dispara automáticamente después de insertar un nuevo registro en la tabla Editorial, y utiliza la función USER () para obtener el nombre del usuario que realizó la acción, la palabra "agregó" para indicar que se agregó un registro, y la función NOW () para obtener la fecha y hora en que se realizó esta acción. Este trigger lo que hará entonces, es insertar un nuevo registro en la tabla "control_de_cambios_librería" con esta información.

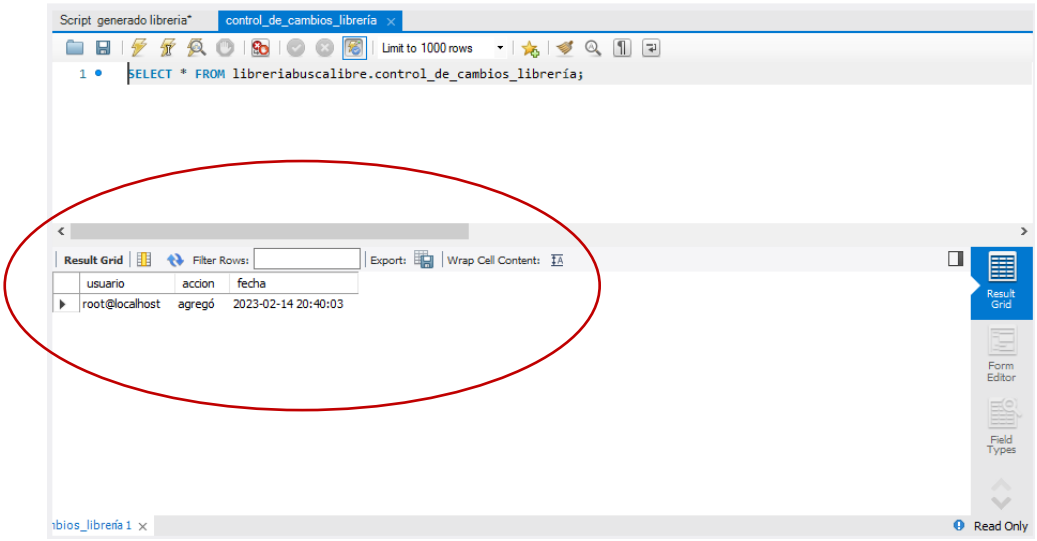
Y , el segundo trigger que creé "eliminar_registro" se disparar también automáticamente después de eliminar un registro de la tabla Editorial, y también utiliza la función USER(), la palabra "eliminó" y NOW() para obtener el nombre del usuario que realizó la acción, la acción realizada y la fecha y hora en que se realizó la acción.

Estos triggers se encargarán de insertar los registros correspondientes en la tabla "control_de_cambios_librería" cada vez que se agregue o elimine un registro en la tabla Editorial.

```sql
-- Crear el primer trigger para insertar en la tabla de control cuando se agregue un registro
DELIMITER //
CREATE TRIGGER insertar_registro AFTER INSERT ON Editorial
FOR EACH ROW
BEGIN
  INSERT INTO control_de_cambios_librería (usuario, accion, fecha)
  VALUES (USER(), 'agregó', NOW());
END//
DELIMITER ;

-- Crear el segundo trigger para insertar en la tabla de control cuando se elimine un registro
DELIMITER //
CREATE TRIGGER eliminar_registro AFTER DELETE ON Editorial
FOR EACH ROW
BEGIN
  INSERT INTO control_de_cambios_librería (usuario, accion, fecha)
  VALUES (USER(), 'eliminó', NOW());
END//
DELIMITER ;
```

Finalmente, para validar el correcto funcionamiento de los triggers creados , verifico insertando datos y observando la tabla que creé para aquellos registros , en mi caso control_de_cambios_libreria:



Para la segunda actividad, relacionada con el script del Hospital , empecé de la siguiente manera:

1. Procedimiento para agregar un procedimiento hospitalario:

creé un procedimiento almacenado en MySQL llamado "AgregarProcedimiento" con dos parámetros de entrada: "p_id" y "p_tipo" y este procedimiento lo utilizo para agregar nuevos procedimientos a la tabla "tb_procedimiento". Con el comando INSERT agregue un nuevo registro a la tabla "tb_procedimiento". El valor de "p_id" se inserta en la columna "id_procedimiento", mientras que el valor de "p_tipo" se inserta en la columna "tipo_procedimiento".

Finalmente con la sentencia CALL hago un llamado al procedimiento:

```
360     -- agregar un procedimiento
361     DELIMITER //
362     CREATE PROCEDURE AgregarProcedimiento(
363       IN p_id VARCHAR(10),
364       IN p_tipo VARCHAR(300)
365     )
366     BEGIN
367       INSERT INTO tb_procedimiento(id_procedimiento, tipo_procedimiento)
368       VALUES(p_id, p_tipo);
369     END //
370     DELIMITER ;
371     DROP procedure AgregarProcedimiento ;
372
373     CALL AgregarProcedimiento ('P6','Revision encias');
374
```

2. Procedimiento para actualizar un procedimiento hospitalario:

Para empezar creé un nuevo procedimiento almacenado llamado "ActualizarProcedimiento y este toma dos parámetros de entrada: "p_id" de tipo VARCHAR de longitud 10 y "p_tipo" de tipo VARCHAR de longitud 300, luego con el comando UPDATE actualizo la columna "tipo_procedimiento" en la tabla "tb_procedimiento" con el valor de "p_tipo" donde el valor de "id_procedimiento" es igual a "p_id"; así:

```
375     -- actualizar un procedimiento
376     DELIMITER //
377     CREATE PROCEDURE ActualizarProcedimiento(
378       IN p_id VARCHAR(10),
379       IN p_tipo VARCHAR(300)
380     )
381     BEGIN
382       UPDATE tb_procedimiento
383       SET tipo_procedimiento = p_tipo
384       WHERE id_procedimiento = p_id;
385     END //
386     DELIMITER ;
387     CALL ActualizarProcedimiento ('P5','sacada cordales');
```

3. Procedimiento para consultar un procedimiento hospitalario:

Para cumplir con lo pedido, creé un procedimiento almacenado en MySQL llamado "ConsultarProcedimiento" con un parámetro de entrada: "p_id", y luego usando el comando SELECT recuperé todas las columnas y sus valores de la tabla "tb_procedimiento" donde el valor de la columna "id_procedimiento" es igual a "p_id".

```
388     -- consultar un procedimiento
389     DELIMITER //
390     CREATE PROCEDURE ConsultarProcedimiento(
391       IN p_id VARCHAR(10)
392     )
393     BEGIN
394       SELECT * FROM tb_procedimiento
395       WHERE id_procedimiento = p_id;
396     END //
397     DELIMITER ;
398     CALL ConsultarProcedimiento ('P5');
```

```
398  CALL ConsultarProcedimiento ('P5');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id_procedimiento	tipo_procedimiento
P5	sacada cordales

4. Procedimiento para borrar un procedimiento hospitalario:

Finalmente, para este último procedimiento creado en la base de datos del Hospital, lo que hice fue crear un nuevo procedimiento almacenado llamado "BorrarProcedimiento". Este procedimiento toma un parámetro de entrada: "p_id" de tipo VARCHAR de longitud 10 y finalmente la lógica del procedimiento: con el comando DELETE eliminé las filas de la tabla "tb_procedimiento" donde el valor de la columna "id_procedimiento" es igual a "p_id".

Esta fue el procedimiento creado:

```
399     -- borrar un procedimiento
400     DELIMITER //
401     CREATE PROCEDURE BorrarProcedimiento(
402       IN p_id VARCHAR(10)
403     )
404     BEGIN
405       DELETE FROM tb_procedimiento
406       WHERE id_procedimiento = p_id;
407     END //
408     DELIMITER ;
409     CALL BorrarProcedimiento ('P5');
```

Finalizando, creé luego los triggers adecuados, pero antes una tabla para almacenar la información de mi interés, en este caso quien agrega o elimina un registro:

```
410
411     -- Crear la tabla "control_de_cambios_hospital"
412 •  CREATE TABLE control_de_cambios_hospital (
413       usuario VARCHAR(50) NOT NULL,
414       accion VARCHAR(50) NOT NULL,
415       fecha DATETIME NOT NULL
416   );
```

Luego, creé los triggers adecuados:

En este ejemplo, el primer trigger que creé "registro_insertar_procedimiento" se disparará después de cada inserción en la tabla "tb_procedimiento", y registra el nombre de usuario actual, la acción realizada ("INSERTAR") y la fecha y hora actual en la tabla "control_de_cambios_hospital" y finalmente el segundo trigger "registro_eliminar_procedimiento" se disparará después de cada eliminación en la tabla "tb_procedimiento", y registra el mismo conjunto de información, pero con la acción "ELIMINAR".

```
417
418     -- Crear el primer trigger para registrar la inserción de un registro en "tb_procedimiento"
419     DELIMITER //
420 •   CREATE TRIGGER registro_insertar_procedimiento
421     AFTER INSERT ON tb_procedimiento
422     FOR EACH ROW
423     BEGIN
424       INSERT INTO control_de_cambios_hospital (usuario, accion, fecha)
425       VALUES (USER(), 'INSERTAR', NOW());
426     END //
427     DELIMITER ;
428
429     -- Crear el segundo trigger para registrar la eliminación de un registro en "tb_procedimiento"
430     DELIMITER //
431 •   CREATE TRIGGER registro_eliminar_procedimiento
432     AFTER DELETE ON tb_procedimiento
433     FOR EACH ROW
434     BEGIN
435       INSERT INTO control_de_cambios_hospital (usuario, accion, fecha)
436       VALUES (USER(), 'ELIMINAR', NOW());
437     END //
438     DELIMITER ;
```