

# Informe Práctica N°05: Procedimientos y Triggers en SQL

Luque Tacora Maria Belen  
Universidad Católica de Santa Maria  
Maria.luque@ucsm.edu.pe

**Resumen.-** En el presente documento se desarrollaron las actividades de la práctica número 05, donde primero se accedió a la base de datos que se trabajó anteriormente, las respectivas tablas y se trabajaron con procedimientos almacenados para que nos permita ejecución más rápida con respecto al gestor de la base de datos y también se trabajó con TIGGER al ejecutarse automáticamente podemos mandar notificaciones al usuario cuando realice acciones que estén prohibidas o dañen el sistema. Por ultimo se concluyo el aprendizaje y bien uso de los procedimientos almacenados y TIGGER que permite ejecución más rápida y de en una base de datos de manera más compleja.

**Palabras Clave-** Tigger, procedimientos, delete, insert, update, begin, end

## I. INTRODUCCIÓN

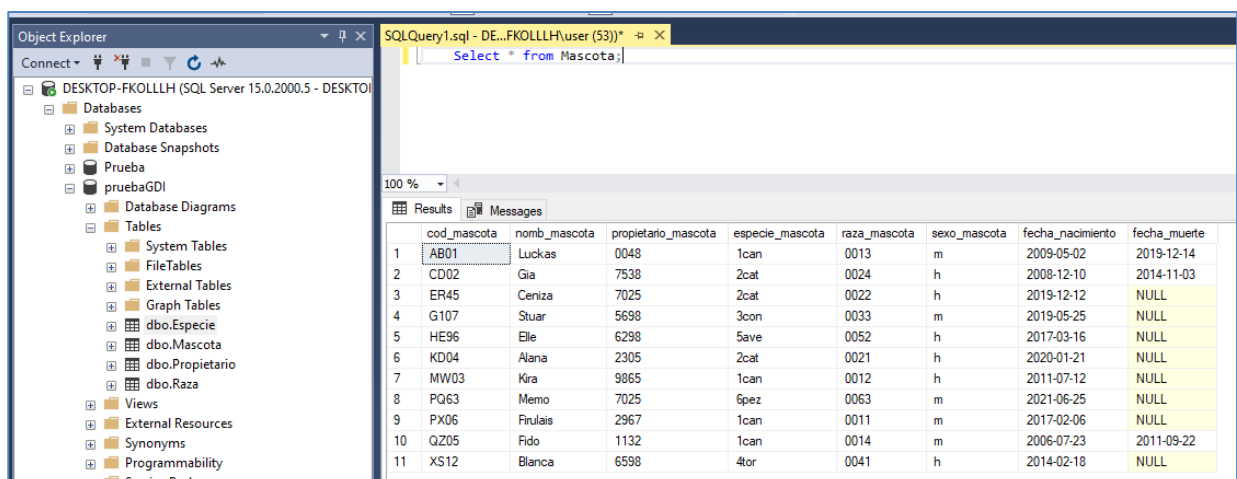
SQL Server es un sistema de gestión de base de datos que tiene un motor de base de datos que controla el almacenamiento, procesamiento y seguridad de los datos [1]. Este sistema fue desarrollado en 1980 y actualmente es uno de los mas usados junto con Oracle Data base y DB2 de IBM, una base de datos es un conjunto de datos que pertenecen a un mismo grupo o contexto y se almacenan sistemáticamente para que se usen posteriormente como para realizar consultas [2]. El procedimiento almacenado (procedimiento almacenado) es un conjunto de declaraciones SQL para completar una función específica, compilada y almacenada en la base de datos, el usuario especifica el nombre del procedimiento almacenado y los parámetros dados (si el procedimiento almacenado tiene parámetros) para llamar y ejecutarlo [3]. TIGGERS son un disparador es un tipo especial de procedimiento de almacenamiento que se ejecuta en respuesta a cierta acción en la tabla, como la inserción, eliminación o actualización de datos. Es un objeto de base de datos que está vinculado a una tabla y se ejecuta automáticamente. No puede invocar activadores de forma explícita. La única forma de hacerlo es realizando la acción requerida en la tabla a la que están asignados. [4]

Por ende, el siguiente documento pretende aumentar el conocimiento del uso de procedimientos de almacenamiento y TIGGERS para realizar consultas mas efectivas y se apliquen en el proyecto elegido por cada grupo, ya que permiten un mejor manejo y uso del gestor de base de datos, para que se tenga una base de datos mas compleja y con consultas que el usuario necesite.

Este documento consta de 4 secciones, la primera, está la introducción al documento; la segunda, son las operaciones o pasos iniciales que emplean para poder empezar a trabajar con SQL SERVER, la tercera, está la descripción de los comandos que se usaron en la base de datos del proyecto RACIEMSA, la función y la ejecución que tendrán con respectivos ejemplos; la cuarta, contiene las conclusiones a las que se llegaron al realizar la práctica y por último en la última sección están las referencias que se usaron para profundizar más en el tema.

## II. OPERACIONES INICIALES

Microsoft SQL SERVER es un entorno de desarrollo en el que podemos administrar cualquier infraestructura SQL, es este podemos administrar, configurar instancias de SQL [5]. Primeramente, se debe establecer la conexión que seria con nuestro servidor local en el este caso, después de realizar la primera operación se creara una base de datos pruebaGDI en el entorno de SQL server, tal como se puede visualizar en la Figure 1., donde se puede visualizar las tablas ya creadas en la practica 04 y los datos que se ingresaron.



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane displays the 'DESKTOP-FKOLLLH (SQL Server 15.0.2000.5 - DESKTOP)' instance. Under 'Databases', the 'pruebaGDI' database is expanded, showing 'Tables' and 'Views'. The 'Tables' folder is expanded, showing 'dbo.Especie', 'dbo.Mascota', 'dbo.Propietario', and 'dbo.Raza'. The 'dbo.Mascota' table is selected. On the right, the 'SQLQuery1.sql - DE...FKOLLLH(user (53))' window shows the query 'Select \* from Mascota;'. Below the query, the 'Results' pane displays the data from the 'Mascota' table in a grid format.

	cod_mascota	nomb_mascota	propietario_mascota	especie_mascota	raza_mascota	sexo_mascota	fecha_nacimiento	fecha_muerte
1	AB01	Luckas	0048	1can	0013	m	2009-05-02	2019-12-14
2	CD02	Gia	7538	2cat	0024	h	2008-12-10	2014-11-03
3	ER45	Ceriza	7025	2cat	0022	h	2019-12-12	NULL
4	G107	Stuar	5698	3con	0033	m	2019-05-25	NULL
5	HE96	Elle	6298	5ave	0052	h	2017-03-16	NULL
6	KD04	Alana	2305	2cat	0021	h	2020-01-21	NULL
7	MW03	Kira	9865	1can	0012	h	2011-07-12	NULL
8	PQ63	Memo	7025	6pez	0063	m	2021-06-25	NULL
9	PX06	Finlala	2967	1can	0011	m	2017-02-06	NULL
10	QZ05	Fido	1132	1can	0014	m	2006-07-23	2011-09-22
11	XS12	Blanca	6598	4tor	0041	h	2014-02-18	NULL

Figura 1: Visualización de las Tablas

### I. Uso de PROCEDURE

Los procedimientos almacenados son: un objeto de base de datos que almacena un programa complejo en la base de datos para que pueda ser llamado por un programa externo.

El procedimiento almacenado se compila y guarda en la base de datos, el usuario puede llamarlo y ejecutarlo especificando el nombre del procedimiento almacenado y los parámetros dados. El procedimiento almacenado es muy simple en pensamiento, es decir, encapsulación y reutilización de código en el nivel de lenguaje SQL de la base de datos. [3]

Con la base de datos pruebaGDI se realizarán procedimientos almacenados:

El primero nos va a permitir listar nombre de las mascotas y sus propietarios por lo que se usara siguiente código en SQL server:

```
create procedure sp_lista_mascota_propietario
as
select m.nomb_mascota 'Mascota',p.nombre 'Dueño'
from mascota m
inner join Propietario p on m.propietario_mascota = p.cod_propietario;
```

Código en SQL server fue el siguiente, lo cual permite realizar consultas más completas:

```
create procedure sp_lista_mascota_propietario
as
select m.nomb_mascota 'Mascota',p.nombre 'Dueño'
from mascota m inner join Propietario p
on m.propietario_mascota = p.cod_propietario;
```

Figura 2: Código del primer PROCEDURE

El segundo PROCEDURE buscara el código de las mascotas que tenga una longitud de 4 con un tipo de dato CHAR:

```
create procedure sp_busca_mascota
@codigo char(4)
as
select * from Mascota where cod_mascota = @codigo;
```

Código en SQL server que busca el código de la mascota:

```
create procedure sp_busca_mascota
@codigo char(4)
as
select * from Mascota where cod_mascota = @codigo;
```

Figura 3: Código del segundo PROCEDURE

El tercer PROCEDURE contara las mascotas que se tienen registradas en total:

```
create procedure sp_cuenta_mascota
@total int output
as
select @total = COUNT(*) from Mascota
return @total
```

Código en SQL server:

```
create procedure sp_cuenta_mascota
@total int output
as
select @total = COUNT(*)
from Mascota
return @total;
```

Figura 4: Código del tercer PROCEDURE

Para poder visualizar las sentencias creadas para los procedimientos almacenados se usa *DECLARE*, como ejemplo en el tercer procedimiento para visualizar lo creado se usa el siguiente código, y se comprueba el correcto funcionamiento del procedimiento:

```
declare @total_mostrar int;
execute sp_cuenta_mascota @total = @total_mostrar output;
print @total_mostrar;
```

El código en SQL server sería el siguiente, con el resultado al ejecutarlo.

```
declare @total_mostrar int;
execute sp_cuenta_mascota @total = @total_mostrar output;
print @total_mostrar;
```

Messages  
11

Figura 5: Código DECLARE el tercer PROCEDURE

Para el primer y segundo procedimiento se usa el siguiente código, que nos permite comprobar que el procedimiento fue implementado de manera correcta:

Código para ejecutar el primer procedimiento:

```
execute sp_lista_mascota_propietario
```

	Mascota	Dueño
1	Luckas	Joshua Infantes
2	Gia	Benita Garcia
3	Ceniza	Valentina Wisnar
4	Stuar	Alejanda Gamero
5	Elle	Joao Ojeda
6	Alana	Paola Lozada
7	Kira	Caroline Bedregal
8	Memo	Valentina Wisnar
9	Finlais	Jose Lopez
10	Fido	Miluska Infantes
11	Blanca	Miluska Infantes

Figura 6: EXECUTE para el primer procedimiento

Código para ejecutar el segundo procedimiento:

```
execute sp_busca_mascota
'AB01'
```

	cod_mascota	nomb_mascota	propietario_mascota	especie_mascota	raza_mascota	sexo_mascota	fecha_nacimiento	fecha_muerte
1	AB01	Luckas	0048	1can	0013	m	2009-05-02	2019-12-14

Figura 7: EXECUTE para el segundo procedimiento

## II. Uso de TRIGGERS

Un disparador es un tipo especial de procedimiento de almacenamiento que se ejecuta en respuesta a cierta acción en la tabla, como la inserción, eliminación o actualización de datos. Es un objeto de base de datos que está vinculado a una tabla y se ejecuta automáticamente. No puede invocar activadores de forma explícita. La única forma de hacerlo es realizando la acción requerida en la tabla a la que están asignados.

EL primer TIGGER que se creo trabaja con la tabla mascotas, donde se notificara en caso el usuario realice modificaciones ya sea insertar, modificar o eliminar dentro de la tabla mascotas y se mostrara un respectivo mensaje.

```
CREATE TRIGGER aviso_urgente
on mascota
after insert, update, delete
as
print 'Se ha modificcado la tabla de mascota .. revisar'
```

```
CREATE TRIGGER aviso_urgente
on mascota
after insert, update, delete
as
print 'Se ha modificcado la tabla de mascota .. revisar'
```

Figura 8: Código del TRIGGER

Código para comprar el funcionamiento del TIGGER:

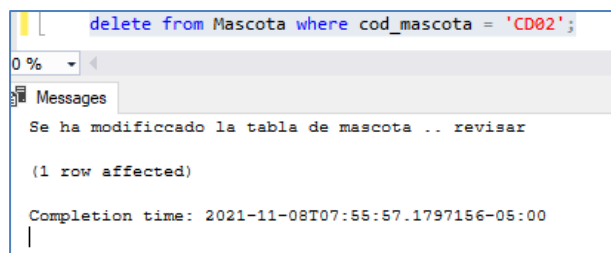


Figura 9: DELETE

### III. OPERACIONES CON LA BASE DE DATOS RACIEMSA

Las operaciones que se realizaron permiten mostrar información que deseamos combinándolas con las de otras tablas, de esta manera se pueden hacer consultas mas complejas y con mejor data. En la generación de un vale de entrada, se añadieron procedimientos que permiten al usuario realizar consultas que, si se usen diariamente de manera más sencilla y fácil de entender, también se hizo de disparadores para advertir al usuario en caso realice la acción que esta no se puede culminar porque podría dañar todo el sistema, también se hizo de uso de una tabla donde almacenara datos o claves que han sido eliminados y otra como historial de actividades.

#### 1. PROCEDIMIENTOS ALMACENADOS

- a. *SIN PARAMETROS*: El procedimiento que se implemento fue para saber cuántas guías de remisión por proveedor hay dentro del almacén RACIEMSA. El código que se uso fue el siguiente:

```
DELIMITER $$
CREATE PROCEDURE ObtenerGuiasPorProv
as
select g.Codigo_guia_remision AS "CODIGO GUIA",
       g.Fecha_de_emision AS "FECHA EMISION",
       p.Razon_social AS "RAZON SOCIAL"
from guia_de_remision g
INNER JOIN proveedor p ON p.Codigo_proveedor = g.Codigo_proveedor
INNER JOIN direcciones d ON d.Codigo_proveedor = p.Codigo_proveedor
end
delimiter ;
```

La figura 8 tiene el código en MySQL y el resultado por pantalla al ejecutar el comando:

The screenshot shows the same SQL code as in Figure 9, followed by a table of results. The table has the following structure and data:

RUC	CODIGO GUIA	FECHA EMISION	RAZON SOCIAL
20543725821	GR.402-0016669	2021-03-20	Distribuidora Cummins Perú SAC.
20543725821	GR.402-0016671	2021-05-23	Distribuidora Cummins Perú SAC.
20543725821	GR.402-0016672	2021-06-18	Distribuidora Cummins Perú SAC.
20100201396	GR.402-0016670	2021-04-17	TRUCASA
20100201396	GR.402-0016673	2021-07-21	TRUCASA

Figura 10: PROCEDURE sin parámetros

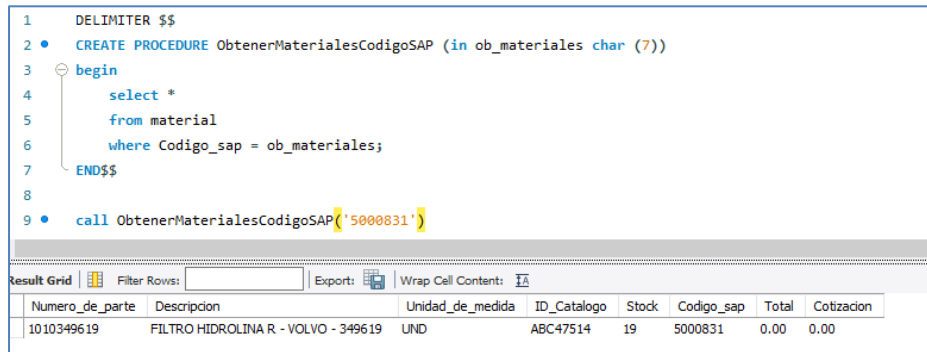
- b. *CON PARAMETROS*: El procedimiento que se implemento fue para buscar materiales dentro del almacén RACIEMSA, pero por el código SAP esto permite que dentro de la empresa se pueda saber si hay aún hay stock disponible y donde esta ubicado, en caso el stock sea se solicita una reposición para el siguiente mes, el proceso también nos permite visualizar la información que tiene como: la descripción, el stock, su unidad de medida, entre otras. El código que se uso fue el siguiente:

```

DELIMITER $$
CREATE PROCEDURE ObtenerMaterialesCodigoSAP (in ob_materiales char (7))
begin
select *
from material
where Codigo_sap = ob_materiales;
END$$
call ObtenerMaterialesCodigoSAP('5000831')

```

Código y resultado por pantalla en MySQL:



The screenshot shows the MySQL Workbench interface. The top pane displays the SQL code for creating and calling a stored procedure. The bottom pane shows the 'Result Grid' with one row of data.

```

1 DELIMITER $$
2 CREATE PROCEDURE ObtenerMaterialesCodigoSAP (in ob_materiales char (7))
3 begin
4     select *
5     from material
6     where Codigo_sap = ob_materiales;
7 END$$
8
9 call ObtenerMaterialesCodigoSAP('5000831')

```

Numero_de_parte	Descripcion	Unidad_de_medida	ID_Catalogo	Stock	Codigo_sap	Total	Cotizacion
1010349619	FILTRO HIDROLINA R - VOLVO - 349619	UND	ABC47514	19	5000831	0.00	0.00

Figura 11: PROCEDURE para obtener materiales a partir del código SAP

- c. **CON VALORES DE RETORNO:** Para este procedimiento se usaron para metros de salida, usando el OUT por lo que se usan las combinaciones ente IN y OUT, el procedimiento que se realizo es para saber cuantas entradas hay un mes a partir de un mismo vale de entrada, esto ayuda a la empresa a tener un mejor control sobre los materiales que se ingresaran al almacén y se pueda corroborar la información en caso haya problemas.

```

DELIMITER $$
DROP PROCEDURE IF EXISTS contar_Entradas$$
CREATE PROCEDURE contar_Entradas(IN valeEntrada VARCHAR(50), OUT total INT UNSIGNED)
BEGIN
SET total = (
SELECT COUNT(*)
FROM entradas
WHERE ID_vale_entrada =valeEntrada);
END
$$
DELIMITER ;
CALL contar_Entradas('13524871', @total);
SELECT @total as 'Total de entradas';

```

Dentro de MySQL el código anterior funciona de la siguiente manera al ejecutar la llamada del procedimiento, para comprobar su correcto funcionamiento.

```

10 DELIMITER $$
11 • DROP PROCEDURE IF EXISTS contar_Entradas$$
12 • CREATE PROCEDURE contar_Entradas(IN valeEntrada VARCHAR(50), OUT total INT UNSIGNED)
13 BEGIN
14     SET total = (
15         SELECT COUNT(*)
16         FROM entradas
17         WHERE ID_vale_entrada =valeEntrada);
18     END
19     $$
20
21 DELIMITER ;
22 • CALL contar_Entradas('13524871', @total);
23 • SELECT @total as 'Total de entradas';
24

```

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

Total de entradas
5

Figura 12: PROCEDURE para contar las entradas por cada vale de entrada

## 2. TIGGERS

Para trabajar con TIGGERS, dentro del proyecto se creo una tabla historial en la cual se registrarán todas las modificaciones que se realizan dentro de las diferentes tablas dentro del almacén RACIEMSA.

Código de la tabla historial:

```

CREATE TABLE `historial` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `accion` VARCHAR(200) NULL,
  `fecha` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`));

```

```

CREATE TABLE `historial` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `accion` VARCHAR(200) NULL,
  `fecha` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`));

```

Figura 13: Código de la tabla historial

### - Insertar

El siguiente TIGGER nos permite saber cuándo se añadió un nuevo material dentro de la tabla material, con la finalidad de posteriormente modificar la tabla ubicaciones para que se tenga un registro correcto, el tener conocimiento de este registro permitirá que el almacén tenga un mejor control sobre los diferentes movimientos que se realicen.

```

DELIMITER //
CREATE TRIGGER aviso_insert_tabla_Material
AFTER INSERT on material
for each row begin
insert into historial(accion)
value (concat('Nuevo Material', NEW.Numero_de_parte));
end//
DELIMITER ;

```

```

DELIMITER //
CREATE TRIGGER aviso_insert_tabla_Material
AFTER INSERT on material
for each row begin
insert into historial(accion)
value (concat('Nuevo Material', NEW.Numero_de_parte));
end//
DELIMITER ;

```

Figura 14: Código del TIGGER para notificar cuando se ingresó un nuevo material

El resultado que se tendrá al realizar un SELECT dentro de la tabla historial será el siguiente:

*INSERT INTO material VALUES ('1676400', 'DEPOSITO AGUA FH 1676400', 'UND', 'ABC47514', '2', '5504167', '2', '0.00');*

	id	accion	fecha
▶	1	Nuevo Material	2021-11-08 00:08:33
	2	Nuevo Material1676400	2021-11-08 00:13:41
*	NULL	NULL	NULL

Figura 15: Resultado de pantalla del primer TIGGER

#### - Modificar

Para este TIGGER, se creó una back cup de proveedor donde se almacenará la información antigua y la que se modifique se registrará en la tabla proveedor principal. El código que se uso fue el siguiente:

```

DELIMITER //
create trigger proveedor_update
before update
on proveedor
for each row
begin
insert into proveedor_back (Codigo_proveedor, Razon_social, RUC) values (old.Codigo_proveedor, old. Razon_social,
old.RUC);
end//
DELIMITER ;

```

```

DELIMITER //
create trigger proveedor_update
before update
on proveedor
for each row
begin
insert into proveedor_back (Codigo_proveedor, Razon_social, RUC) values (old.Codigo_proveedor, old. Razon_social, old.RUC);
end//
DELIMITER ;

```

Figura 16: Código en MySQL

Para comprobar el correcto funcionamiento del TIGGER se realizó la siguiente modificación en tabla proveedor y en la tabla proveedor\_back se guardará el anterior cambio en caso sea necesario.

*update proveedor set Razon\_social='TRUCASA' where Codigo\_proveedor='2060018016';*

	id	Codigo_proveedor	Razon_social	RUC
▶	1	2060018016	Automotriz Andina S.A.	20100201396
*	NULL	NULL	NULL	NULL

Figura 17: Datos anteriores del RUC 2060018016

	Codigo_proveedor	Razon_social	RUC
▶	2060018016	TRUCASA	20100201396
	2060018017	Distribuidora Cummins Perú SAC.	20543725821
	2060018018	AUTRISA	20100202396

Figura 18: Nuevos Datos

#### - Borrar

Para el TIGGER, en este caso usamos *BEFORE*, en caso el usuario desee eliminar el catálogo del almacén, no se permitirá realizar tal acción, por lo que le saldrá un mensaje “No se puede el catálogo”. El código para realizar el *TIGGER* fue el siguiente:

```

DELIMITER //
CREATE TRIGGER eliminar_Catalogo
BEFORE DELETE on catalogo
for each row begin
SIGNAL sqlstate '02000' SET message_text = 'No se puede eliminar el catalogo';
end//
DELIMITER ;

```

En el MySQL se realizo de la siguiente manera:

```
DELIMITER //
CREATE TRIGGER eliminar_Catalogo
BEFORE DELETE on catalogo
for each row begin
    SIGNAL sqlstate '02000' SET message_text = 'No se puede eliminar el catalogo';
end//
DELIMITER ;
```

Figura 19: TIGGER que no permite que se elimina una solicitud

Para verificar se intentó eliminar una solicitud y en la siguiente figura se puede observar el resultado que se tuvo:

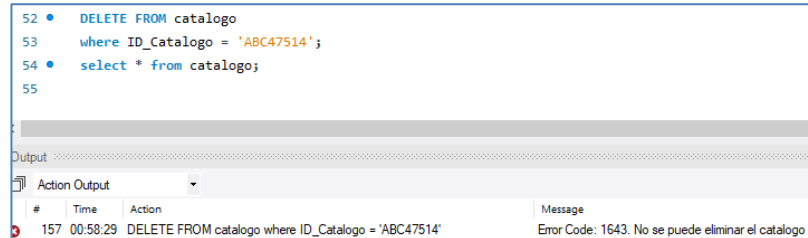


Figura 20: Verificación del correcto funcionamiento del TIGGER

#### - Insertar

El siguiente TIGGER fue para la tabla proveedor y se tenga registrado cuando se realice la acción de añadir un nuevo proveedor, y esta a su vez se almacene dentro de historial. El código que se uso fue el siguiente:

El código en MySQL y la captura de pantalla fueron los siguientes:

```
DELIMITER //
CREATE TRIGGER aviso_insert_tabla_Proveedor
AFTER INSERT on proveedor
for each row begin
    insert into historial(accion)
    value (concat('Nuevo Proveedor: ', NEW.Codigo_proveedor));
end//
DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER aviso_insert_tabla_Proveedor
AFTER INSERT on proveedor
for each row begin
    insert into historial(accion)
    value (concat('Nuevo Proveedor: ', NEW.Codigo_proveedor));
end//
DELIMITER ;
```

Figura 21: TIGGER en MySQL para proveedor

	id	accion	fecha
▶	1	Nuevo Material	2021-11-08 00:08:33
	2	Nuevo Material1676400	2021-11-08 00:13:41
	3	Nuevo Proveedor: 2060018018	2021-11-08 00:29:01
✱	NULL	NULL	NULL

/ Figura 22: Registro del nuevo proveedor registrado

#### - Modificar

El siguiente TIGGER se realizo con la finalidad de almacenar los cambios que se realicen después de registros del proveedor, en caso se hayan cometido errores;

```
DELIMITER //
create trigger proveedor_updateAFT
after update
on proveedor
for each row
```



```

begin
insert into proveedor_back values (current_user(), concat(' UPDATE de: ', old.RUC, ' ', now()));
end//
DELIMITER ;

insert into proveedor values ('2060018018', 'Distribuidora Cummins Perú SAC.', '20543725821');
select * from proveedor

DELIMITER //
create trigger proveedor_updateAFT
after update
on proveedor
for each row
begin
insert into proveedor_back values (current_user(), concat(' UPDATE de: ', old.RUC, ' ', now()));
end//
DELIMITER ;

insert into proveedor values ('2060018018', 'Distribuidora Cummins Perú SAC.', '20543725821');
select * from proveedor;
update proveedor
set RUC = '27100202376' where Codigo_proveedor = '2060018019';

```

Figura 23: UPDATE con AFTER

#### - Borrar

Se notificará al usuario que ha eliminado un material, y se guardara la información del material. El código fue el siguiente:

```

DELIMITER //
CREATE TRIGGER eliminar_Material
AFTER DELETE on material
for each row
insert into historial (accion) value (current_user(), concat('Se elimino lo siguiente:', old.Numero_de_parte, ' ',
old.Descripcion, ' ', old.Codigo_sap, ' ', old.Unidad_de_medida, ' ',
old.Stock, ' ', old.Total, ' ', old.Cotizacion, ' ', now()));
end//
DELIMITER ;

```

```

DELIMITER //
CREATE TRIGGER eliminar_Material
AFTER DELETE on material
for each row
insert into historial (accion) value (current_user(), concat('Se elimino lo siguiente:', old.Numero_de_parte, ' ',
old.Descripcion, ' ', old.Codigo_sap, ' ', old.Unidad_de_medida, ' ',
old.Stock, ' ', old.Total, ' ', old.Cotizacion, ' ', now()));
end//
DELIMITER ;
DELETE FROM material
where Codigo_sap = '5000831';
select * from material;

```

Figura 24: DELETE con AFTER

## IV. CONCLUSIONES

- SQL server es un sistema de gestor de base de datos que es usado por muchos desarrolladores de bases de datos, ya que permite el manejo de información para algunos más rápido, también se puede realizar operación más rápido sin la necesidad de tener que usar un QUERY de por medio en caso sea añadir data o modificar el tipo de valor dentro de una tabla.
- Los procesos de almacenamiento, permiten realizar mejores consultas al usuario, ya que el usuario no tiene por qué escribir tanto código para que busque información dentro de la base de datos.
- Los disparadores o TIGGER, son utiles pues permite guardar información, en caso el usuario la borre, o que una sentencia dentro de BEGIN y END se cumpla para cuando se desee modificar, insertar y eliminar.

## REFERENCIAS

- [1] ComputerWeekly.com, «Microsoft SQL Server,» [En línea]. Available: <https://www.computerweekly.com/es/definicion/Microsoft-SQL-Server>. [Último acceso: 17 Oct 2021].

- [2] NEXTECH, «¿Qué es SQL Server? y ¿Para qué sirve?,» [En línea]. Available: <https://nextech.pe/que-es-sql-server-y-para-que-sirve/>. [Último acceso: 17 Oct 2021].
- [3] E. Lazaro, «Cómo crear y usar un procedimiento almacenado MySQL,» 14 Abril 2019. [En línea]. Available: <https://www.neoguias.com/procedimientos-almacenados-mysql/>. [Último acceso: 07 Noviembre 2021].
- [4] J. J. S. Hernández, «Unidad 12. Triggers, procedimientos y funciones en MySQL,» 2021. [En línea]. Available: <https://josejuansanchez.org/bd/unidad-12-teoria/index.html#triggers-1>. [Último acceso: 7 Noviembre 2021].
- [5] M. Ignite, «Download SQL Server Management Studio (SSMS),» [En línea]. Available: <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>. [Último acceso: 17 Oct 2021].