Tecnológico Nacional de México
Instituto Tecnológico de Mexicali

Ingeniería en Sistemas Computacionales

Taller de Base de Datos

Reporte de práctica. Scripts relacionados al proyecto.

Barba Navarro Luis Rodrigo, 20490687
Gurrola Bernal Johan Antonio, 20490703
Lira López Alejandro, 16491160
Pérez García Sofía, 20491083

No. Equipo: 4
Grupo: 3 p.m - 4 p.m, CZA 5
Profesor: Carlos Alberto López Castellanos

Mexicali, Baja California a viernes, 09 de diciembre de 2022.

Índice.

I. Introducción.

En la actualidad, fundamentalmente en las empresas que van saliendo día con día, muchas decisiones se toman en función de sus prioridades actuales. Hasta aquí todo ocurre de manera correcta y sin desvíos en cuestión de decisiones. Sin embargo, hay movimientos dentro de la empresa aparentemente sin importancia que, en el futuro próximo de la empresa, resultará de manera algo inesperada ser bastante importantes. Con esto nos referimos a la vital ocurrencia con respecto a dónde almacenar los datos tanto empresariales como del usuario.

Generalmente, debido a su importancia comúnmente infravalorada, se elige la forma más rápida, la cual es almacenar datos en archivos de hojas de cálculo. Pero esto resulta en algo ineficiente y poco productivo para los empleados de la empresa; siendo algo hasta desgastante. Los sistemas de archivos simples no están hechos para eso. Por lo tanto, no es buena práctica esperar demasiado antes de introducir un sistema de base de datos dentro de una empresa, de lo contrario, la actualización tendría muchos efectos negativos a corto y mediano plazo. Cuantos más datos haya almacenado en archivos, más complejo y costoso será el desarrollo de la base de datos. Además, toma más tiempo, lo que en algunos casos podría causar mucho daño financiero.

Durante el transcurso del semestre, contemplamos la relevancia de las bases de datos para las empresas; estas nos permiten almacenar grandes volúmenes de datos en un solo lugar. Los conocimientos adquiridos fueron desde cómo obtener los requerimientos funcionales del cliente para desarrollar el modelo entidad-relación y tomar decisiones sobre cómo modelar la base de datos, hasta el manejo de acceso de usuarios y administración de sus respectivos privilegios o roles, implementación y seguimiento de transacciones, procedimientos almacenados, manejadores y por último, la configuración de los disparadores, todo esto anteriormente mencionado dentro de una base de datos, que en este caso, se trabajó durante todo el semestre con la base de datos del proyecto nombrada 'maquina_expendedora'.

En el presente reporte de práctica, se pretende desarrollar y resolver cuatro ejercicios que tienen como objetivo implementar todo el conocimiento que hemos aprendido en el transcurso del semestre; sobre todo en el manejo de transacciones, procedimientos almacenados, manejadores y disparadores, esto con la finalidad de reconocer la importancia que es tener un buen conocimiento relacionado a las bases de datos en el mundo laboral, por la alta demanda de ingenieros, donde se solicita que tengan dominio fundamental sobre este tema.

II. Justificación.

La razón principal por la cual se realizará este reporte de práctica es para reconocer la gran importancia que es tener un buen conocimiento relacionado a la administración de las bases de datos, sobre todo en la actualidad donde la mayoría de empresas optan por implementar un sistema de información basado en base de datos que permitan suplir con la necesidad de almacenar grandes volúmenes de información por parte de sus empleados y usuarios.

Hoy en día, los sistemas de bases de datos son esenciales para los negocios ya que comunican información relacionada con sus transacciones de ventas, inventario de productos, perfiles de clientes y actividades relacionadas a la mercadotecnia. Sin un lugar en específico donde almacenar toda esta información, es muy probable que no se tenga una idea clara de lo que está ocurriendo dentro del negocio.

III. Objetivos.

A. Reconocer la importancia de la implementación de una base de datos como solución a los problemas de la empresa en cuestión de almacenamiento de información por parte de los usuarios y empleados.
B. Contemplar los conocimientos que un ingeniero de sistemas computacionales debe dominar para la administración de una base datos, que le permitan mejorar su competitividad en el aspecto laboral.
C. Investigar soluciones a los problemas planteados en la práctica donde se implementen los procedimientos almacenados, disparadores, manejadores y transacciones sobre la base de datos del proyecto.
D. Reconocer las diferentes formas de implementar una solución que se pueden llegar a presentar al momento de resolver un problema con relación a la base de datos del proyecto.

IV. Desarrollo.

A. Ejercicio 1.

Escriba un trigger que al dar de alta una tarjeta nueva a un cliente, deshabilite todas las tarjetas que tenga y transfiera el saldo a la nueva tarjeta.

Sentencia SQL

```
-- Procedimiento que asigna tarjeta a un cliente.
DROP PROCEDURE IF EXISTS assign_card;
DELIMITER //
CREATE PROCEDURE assign_card
(
    IN id_tarjeta INT,
    IN id_cliente INT
)
BEGIN
    -- Tarjeta duplicada.
    DECLARE EXIT HANDLER
    FOR 1062
    SELECT 'Identificador de tarjeta duplicado.';
    -- Cliente no encontrado.
    DECLARE EXIT HANDLER
    FOR 1452
    SELECT 'Identificador de cliente no encontrado.';
    SET @puntos = 0;
    SELECT count(*) INTO @conteo FROM tarjeta WHERE tarjeta.id_cliente = id_cliente AND estatus = 'Habilitada';
    IF @conteo > 0
        THEN CALL get_points(id_tarjeta, id_cliente, @puntos);
    END IF;
    INSERT tarjeta VALUES (id_tarjeta, id_cliente, CURRENT_DATE(), @puntos, 'Habilitada');
    SELECT 'Tarjeta agregada satisfactoriamente.';
END //

DELIMITER ;
```

```sql
-- Procedimiento obtiene los puntos de las tarjetas habilitadas del cliente.
DROP PROCEDURE IF EXISTS get_points;
DELIMITER //
CREATE PROCEDURE get_points
(
IN id_tarjeta INT,
IN id_cliente INT,
OUT puntos INT
)
BEGIN
    SELECT sum(tarjeta.puntos)
    INTO puntos
    FROM tarjeta
    WHERE estatus = 'Habilitada' AND tarjeta.id_cliente = id_cliente;

    UPDATE tarjeta
    SET tarjeta.puntos = 0, tarjeta.estatus = 'Deshabilitada'
    WHERE tarjeta.id_cliente = id_cliente AND tarjeta.estatus = "Habilitada";
END //

DELIMITER ;
```

Resultado

// Se encarga de agregar la nueva tarjeta

Id_tarjeta / id_cliente

```
CALL assign_card(23, 20490689);
```

Result Grid | Filter Rows:

	Tarjeta agregada satisfactoriamente.
▶	Tarjeta agregada satisfactoriamente.

Validación de asignación de la nueva tarjeta

```
select * from tarjeta where id_tarjeta=23;
```

	id_tarjeta	id_cliente	fecha_expedicion	puntos	estatus
▶	23	20490689	2022-12-08	1858	Habilitada
*	NULL	NULL	NULL	NULL	NULL

Explicación

En este se usan dos procedimientos almacenados, uno para la asignación de una nueva tarjeta y otro para la sumatoria de todos los puntos del cliente de la nueva tarjeta, para guardarlo en una variable global que se usará para transferir los puntos a la nueva tarjeta.

En el primer procedimiento se declaran los handler para detectar si la id de la tarjeta saldrá duplicada o si el cliente a quien se le hará la tarjeta, luego realizamos un conteo para verificar si el cliente tiene tarjetas habilitadas las cuales tendríamos que sacar puntos de, en caso de que si se llama el segundo procedimiento que se encarga de sumar los puntos de todas las tarjetas que tiene y simultáneamente deshabilitarlas.

Finalmente, inserta la tarjeta nueva con una id, la id del cliente, la fecha de ahora, la sumatoria de los puntos a transferir y estableciendo su estatus como Habilitada.

B. Ejercicio 2.

Escriba un trigger que al solicitar un alta de un producto en un determinado slot de una maquina valide que no se sobrepase de la capacidad, que ese slot exista y que sea el mismo producto, en caso de ocurrir lance un error. Considere que la capacidad de los slots es de 30 productos y que son 80 slots por máquina

Sentencia SQL

```
-- Procedimiento almacenado para mostrar inventario de una máquina expendedora en específico;
DROP PROCEDURE IF EXISTS show_inventory_vending
DELIMITER //
CREATE PROCEDURE show_inventory_vending
(
    IN id_maquina VARCHAR(4)
)
BEGIN
    SELECT *
    FROM inventario_maquina
        INNER JOIN producto using (id_producto)
            INNER JOIN categoria using (id_categoria)
    WHERE inventario_maquina.id_maquina = id_maquina;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE add_product
(
    IN id_maquina VARCHAR(4),
    IN id_producto INT,
    IN numero_slot VARCHAR(3),
    IN cantidad INT
)
BEGIN
    IF
    (
        SELECT count(*)
        FROM inventario_maquina
        WHERE inventario_maquina.numero_slot = numero_slot
        AND inventario_maquina.id_maquina = id_maquina
    ) > 0
    THEN
        IF
        (
            SELECT count(*)
            FROM inventario_maquina
            WHERE inventario_maquina.numero_slot = numero_slot
            AND inventario_maquina.id_producto = id_producto
            AND inventario_maquina.id_maquina = id_maquina
        ) > 0
        THEN
            UPDATE inventario_maquina
            SET inventario_maquina.cantidad = inventario_maquina.cantidad + cantidad
            WHERE inventario_maquina.id_maquina = id_maquina
            AND inventario_maquina.id_producto = id_producto
            AND inventario_maquina.numero_slot = numero_slot;
        ELSE
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error. Producto agregado no coincide con el producto que se encontraba anteriormente.';
        END IF;
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error. Número de slot no existe';
    END IF;
END //
DELIMITER  ;
```

```
-- Disparador que valida que no se sobrepase de la capacidad, que el slot exista y que sea el mismo producto;
DROP TRIGGER IF EXISTS verify_capability_BU;
DELIMITER //
CREATE TRIGGER verify_capability_BU BEFORE UPDATE on inventario_maquina
FOR EACH ROW
BEGIN
    IF new.cantidad > 30 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error. Se excedió la capacidad del slot en la máquina expendedora.';
    END IF;
END //
DELIMITER ;
```

Resultado

```
-- Demostración
CALL show_inventory_vending('4010');
CALL add_product('4010', 159001, 'A02', 8);
```

Action Output

#	Time	Action	Message
1	23:08:59	CREATE PROCEDURE show_inventory_vending (IN id_maquina VARCHAR(4)) BEGIN S...	0 row(s) affected
2	23:09:10	CREATE PROCEDURE add_product (IN id_maquina VARCHAR(4), IN id_producto INT, IN...	0 row(s) affected
3	23:10:03	CREATE TRIGGER verify_capability_BU BEFORE UPDATE on inventario_maquina FOR E...	0 row(s) affected
4	23:10:27	CALL show_inventory_vending('4010')	50 row(s) returned

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

id_categoria	id_producto	id_maquina	numero_slot	cantidad	precio_unitario	nombre_producto	descripcion_producto	nombre_categoria	descripcion_categoria
2010	159001	4010	A01	6	36	Agua Bonafont	por 100ml - Calorias: 0kcal | Grasa: 0,00g	Botellas de agua	Agua potable envasada en
2010	159002	4010	A02	6	40	Agua Ciel	por 100ml - Calorias: 0kcal | Grasa: 0,00g	Botellas de agua	Agua potable envasada en
2010	159003	4010	A03	4	36	Agua Epura	por 100ml - Calorias: 0kcal | Grasa: 0,00g	Botellas de agua	Agua potable envasada en
2011	159004	4010	A04	9	29	Bebida Monster	por 100ml - Calorias: 49kcal | Grasa: 0,00g	Bebidas energeticas	Bebida envasada estimulan
2011	159005	4010	A05	9	21	Bebida Boost	por 1 porción (200ml) - Calorias: 102kcal | Gras...	Bebidas energeticas	Bebida envasada estimulan
2011	159006	4010	A06	8	28	Bebida Red Bull	por 100ml - Calorias: 45kcal | Grasa: 0,00g	Bebidas energeticas	Bebida envasada estimulan
2014	159007	4010	A07	5	53	Minerale Epura	por 1 botella (600ml) - Calorias: 24kcal | Grasa: ...	Refrescos mineral	Refresco envasado con car
2014	159008	4010	A08	0	31	Peñafiel	por 100ml - Calorías: 0kcal | Grasa: 0,00g	Refrescos mineral	Refresco envasado con car
2014	159009	4010	B01	3	55	Perrier	por 100ml - Calorías: 0kcal | Grasa: 0,00g	Refrescos mineral	Refresco envasado con car

Explicación

Se crearon 2 procedimientos, el primer procedimiento se llama show_inventory_vending el cual tiene una variable llamada id_maquina la cual tiene su parámetro de entrada, cuando comienza el procedimiento muestra todos los campos de la tabla inventario_maquina siempre y cuando el id_maquina sea el que proporcionamos en el procedimiento.

El segundo procedimiento se llama add_product el cual cuenta con varias variables id_ maquina, id_producto, numero_slot, cantidad todas tienen sus paramentos de entrada comienza el procedimiento y comienza una sentencia condicional en la cual cuenta todas las filas en la tabla inventario_maquina en este caso, donde el numero_slot = A01 y inventario_maquina = 4010 y sea mayor a 0. En el momento en el que la condición y la expresión llegan a coincidir, se devuelve la expresión mencionada en la cláusula THEN en la cual realiza varias validaciones para evitar que se coloque un producto en un slot que no

corresponde o evitar que se coloque producto en un slot no existente. Si se pasan todas las condiciones, el programa actualizará la cantidad de producto en el slot solicitado y en la máquina solicitada.

El primer Trigger se llama verify_capability_BU en el cual se especifica que antes de la actualización en inventario_maquina, el Trigger se disparará cada vez que se realizan operaciones sobre cada fila de la tabla, esto nos ayuda a cuando se presente algún error relacionado a la capacidad del slot nos permitirá validar que sea correcto, en caso de no estar dentro de los parámetros adecuados el cambio no se realizará

C. Ejercicio 3.

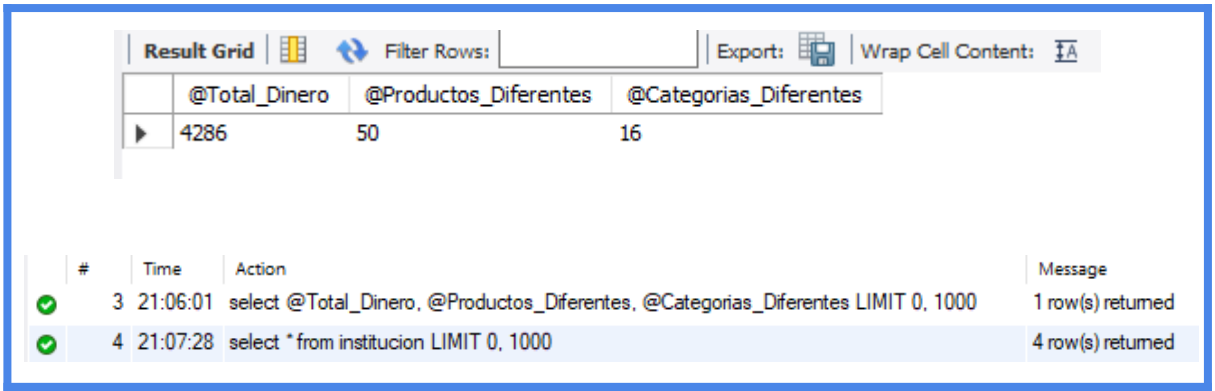
Escriba un procedimiento que realice un inventario de productos de todas las maquinas de una institución en particular, debe regresar el total en dinero de todo el producto, la cantidad de productos diferentes que se venden y la cantidad de categorías diferentes que se venden

Sentencia SQL

```
-- Procedimiento almacenado que recibe un identificador de una institución, y muestra el total de dinero en cuestión de
-- productos, cantidad de productos diferentes que se venden y la cantidad de categorías diferentis que se venden.
drop procedure if exists inventario;
delimiter //
create procedure inventario
(
    in id_inst int,
    out Total_Dinero int,
    out Productos_Diferentes int,
    out Categorias_Diferentes int
)
begin
    if (select count(*) from institucion where id_institucion = id_inst) = 0 then
        begin
            signal sqlstate '45000'
            set message_text = 'Error. Institución no encontrada';
        end;
    else
        /*Dinero total*/
        select sum(precio_unitario) into Total_Dinero
        from inventario_maquina
        inner join maquina using (id_maquina)
        where id_institucion=id_inst;

        /*Cantidad diferente de productos*/
        select count(distinct producto.id_producto) into Productos_Diferentes
        from producto
        inner join inventario_maquina using (id_producto)
            inner join maquina using (id_maquina)
        where id_institucion=id_inst;

        /*Cantidad de categorías*/
        select count(distinct id_categoria) into Categorias_Diferentes
        from producto
        inner join inventario_maquina using (id_producto)
            inner join maquina using (id_maquina)
        where id_institucion=id_inst;

    end if;
end //
delimiter ;
```

Resultado

```
-- Demostración
call inventario(1040, @Total_Dinero, @Productos_Diferentes, @Categorias_Diferentes);
select @Total_Dinero, @Productos_Diferentes, @Categorias_Diferentes;
```



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

	@Total_Dinero	@Productos_Diferentes	@Categorias_Diferentes
▶	4286	50	16

#	Time	Action	Message
3	21:06:01	select @Total_Dinero, @Productos_Diferentes, @Categorias_Diferentes LIMIT 0, 1000	1 row(s) returned
4	21:07:28	select * from institucion LIMIT 0, 1000	4 row(s) returned

Explicación

Se creó un procedimiento llamado inventario en el cual se declaró los parámetros de los campos de salida ,se ejecuta el procedimiento y realiza una validación de la institución para corroborar que si existe la institución en caso de no existir arroja un mensaje de error indicando "Error Institución" y se termina el procedimiento , en caso contrario procede a contar el dinero total en producto, cantidad de diferentes productos, cantidad de categorías y procede a terminar el procedimiento inventario, Posteriormente se realizaron las comprobaciones correspondientes, se utilizó como ejemplo id_institucion 1040 que le pertenece a universidad Xochicalco Mexicali.

D. Ejercicio 4.

Escriba un store procedure que reciba como parámetro un id de tarjeta, un id de producto y una cantidad, y que:

- **Actualice el inventario de productos de la máquina**
- **El saldo de la tarjeta**
- **Valide que la tarjeta tenga el saldo suficiente para la venta, en caso de no tenerlo debe lanzar un error**
- **Utilice transacciones**

Sentencia SQL

```
delimiter //
create procedure buy_product
-- Parámetros
(
    in id_maquina_V int,
    in id_tarjeta_V int,
    in id_producto_V int,
    in buying int
)
-- Inicio del procedimiento
begin
    declare points_to_pay, M_Verify, T_Verify, P_Verify int;
    declare exit handler for sqlstate '45000' rollback;

    set M_Verify = 0;
    set T_Verify = 0;
    set P_Verify = 0;

    select (buying * precio_unitario) into points_to_pay
    from inventario_maquina
    where id_maquina = id_maquina_V
    and id_producto = id_producto_V;

    start transaction;
    select count(*) into M_Verify from inventario_maquina where id_maquina = id_maquina_V;
    select count(*) into P_Verify from inventario_maquina where id_maquina = id_maquina_V and id_producto = id_producto_V;
    select count(*) into T_Verify from tarjeta where id_tarjeta = id_tarjeta_V;
```

```
    if M_Verify > 0 and P_Verify > 0 and T_Verify > 0 then

    set @message = 'ERROR: Insuficient product.';
    update inventario_maquina
    set cantidad = cantidad - buying
    where id_maquina = id_maquina_V
    and id_producto = id_producto_V;


    set @message = 'ERROR: Insuficient funds.';
    update tarjeta
    set puntos = puntos - points_to_pay
    where id_tarjeta = id_tarjeta_V;

    end if;

    set @message = 'Transaction complete, enjoy your product!';

    commit;
end //
delimiter ;
```

```
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

-- Trigger for the card.
drop trigger no_points_below_zero_BU;
delimiter //
create trigger no_points_below_zero_BU before update on tarjeta
for each row
begin
    if new.puntos < 0 then
        signal sqlstate '45000'
        set message_text = 'ERROR: Insuficient funds.';
    end if;
end //
delimiter ;

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

-- Trigger for the products in the machine.
drop trigger no_product_below_zero_BU;
delimiter //
create trigger no_product_below_zero_BU before update on inventario_maquina
for each row
begin
    if new.cantidad < 0 then
        signal sqlstate '45000'
        set message_text = 'ERROR: Insuficient product.';
    end if;
end //
delimiter ;
```

Resultado

```
set @message = '';
call buy_product(4010, 10020512, 159001, 2);
select @message;
```

	@message
▶	Transaction complete, enjoy your product!

Explicación

Nuestro proceso primero se encarga de declarar las variables y el handler necesario para encontrar errores, las variables para ver si la máquina, producto y la tarjeta existen y el handler para cuando el producto o los puntos en la tarjeta son insuficientes.

Después determinamos los puntos necesarios para llevar a cabo la compra y lo guardamos en una variable y finalmente empezamos una transacción ya que se verifique la existencia de todo lo necesario para la compra.

Los siguientes updates pueden darnos errores, así que tenemos predecir el error más posible a pasar, para el primer update es en el que el producto sea insuficiente (lo cual es verificado por nuestro trigger que se activa antes del update) y para el segundo es para verificar que hayan suficientes puntos para comprarlo (igual que el anterior, avienta un error por medio del trigger que verifica si se tienen suficientes).

Finalmente nada más mandamos el mensaje de que fue exitosa la transacción, o en caso contrario, el tipo de error que se presentó.

V. Conclusión y recomendaciones.

Con base a lo anteriormente mencionado, se puede concluir que mediante la realización de los ejercicios planteados pudimos aprender el uso adecuado de los disparadores, procedimientos almacenados y manejadores, teniendo en cuenta su funcionamiento y las condiciones de uso. Los disparadores se pueden usar para insertar, actualizar o borrar registros de otras tablas o incluso otras bases de datos, cuando se realiza un evento en la tabla propuesta en el disparador, donde la principal diferencia entre los disparadores y procedimientos almacenados, es la ventaja que tiene que los disparadores se ejecutan de manera automática cuando se realiza cierto evento.

Asimismo, un procedimiento almacenado es aquel procedimiento que debe ser invocado para ejecutarse, puede recibir parámetros y devolver parámetros después de la operación. Además, se puede manejar cualquier tabla, realizar operaciones con ellas y realizar iteraciones de lectura o escritura. Por último, se vio que los manejadores especifican qué pueden hacer ante la presencia de un error. Si una de estas condiciones de error ocurre, las instrucciones dentro del manejador especificado se ejecutan. Esto nos ayuda a prever algún error y mandar un mensaje dependiendo de cuál error sea.

No obstante, mediante la realización de los ejercicios no hubo problemas excepto en el primer ejercicio, debido a que nos pedía implementar disparadores pero ocasionó un problema el cúal es que no se puede actualizar la tabla del procedimiento almacenado o disparador cuando ya está utilizada por la instrucción que invocó ese procedimiento almacenado o disparador. Para solucionar el error, simplemente se acondicionó la transferencia de puntos de las demás tarjetas habilitadas del cliente, en otro procedimiento almacenado, y se mandaba llamar cuando el procedimiento detectará si el cliente propuesto tiene una o más tarjetas habilitadas con puntos.

En definitiva, la elaboración de los ejercicios propuestos nos sirvió para practicar más sobre la buena implementación de los procedimientos almacenados, disparadores y manejadores, sobre todo para no atrofiar nuestro conocimiento; esperando que esto nos pueda servir a futuro en próximos proyectos que requieran manejo de bases de datos.

Recomendaciones.

- Se debe de considerar que la principal diferencia entre los disparadores y procedimientos almacenados; Los disparadores son procedimientos que se ejecutan automáticamente, cuando se produce un evento sobre el que se quiere trabajar. Para esto existen tres tipos de eventos que pueden disparar un disparador: INSERT, DELETE y UPDATE.
- El disparador se programa para realizar una tarea determinada que se debe hacer siempre que se produzca uno de los eventos antes mencionados. No requiere intervención humana.

- Un procedimiento almacenado es un código SQL preparado que puede recibir valores, procesarlos y dar un resultado, por lo que el código puede reutilizarse una y otra vez; siendo una buena práctica y mejora la eficiencia.
- También se puede enviar como parámetros, variables de entorno a un procedimiento almacenado, de modo que el procedimiento pueda actuar en función de los valores de parámetro que se pasan, y puedan usarse dichas variables de manera externa.
- Es importante que la tabla que tiene asignada el disparador para activarse cuando se realice un evento, no modifique esa misma tabla dentro del disparador ya que podría causar problemas.