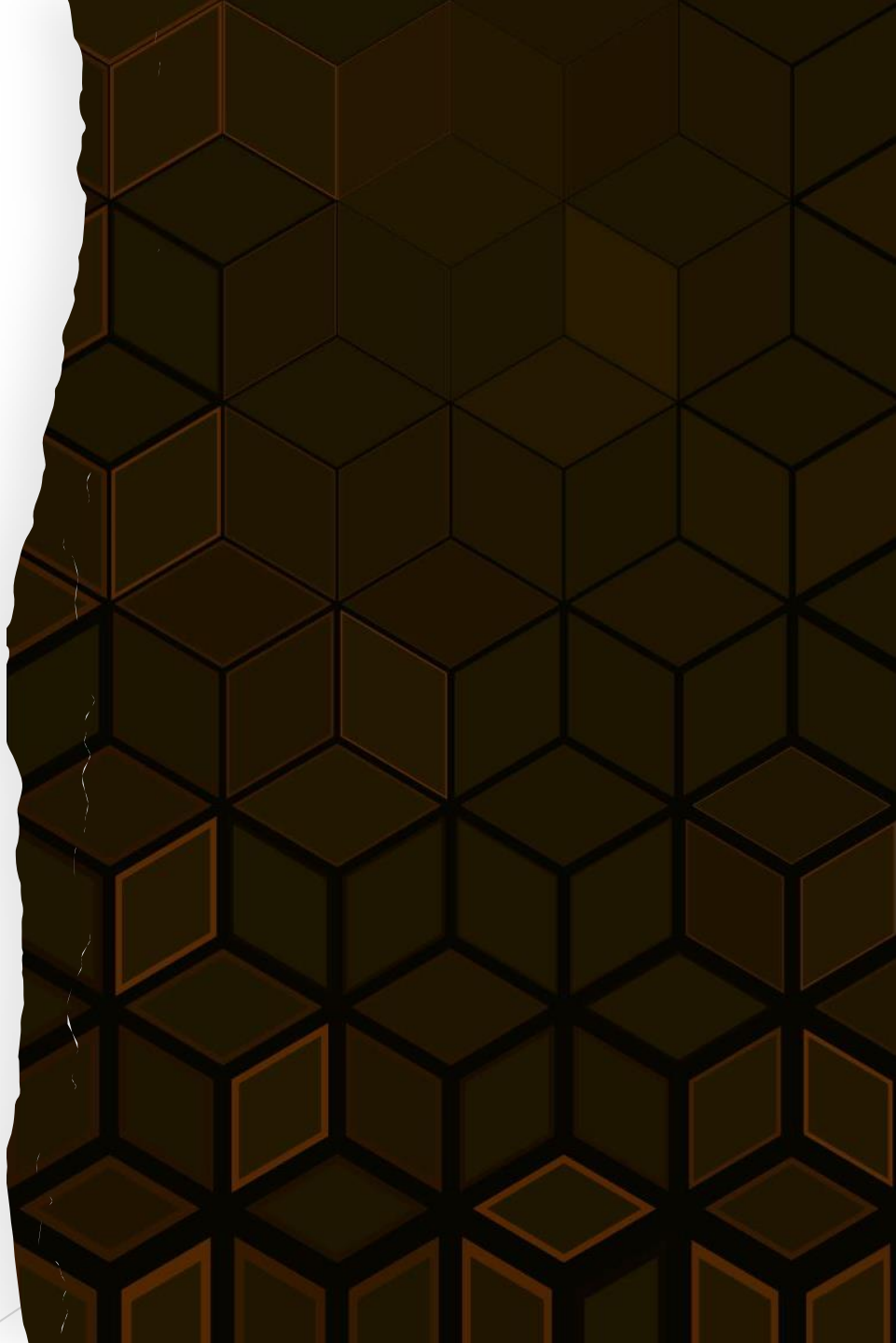


# Triggers, cursores y excepciones



¿Qué sabemos  
hacer?

---

Sabemos hacer  
procedimientos y funciones

---

Sabemos crear bucles y  
iteraciones

---

Sabemos ejecutar todo eso  
y mezclarlo con sentencias  
SQL

# Triggers

Los triggers o disparadores, son procedimientos que se ejecutan de forma automática cuando pasa algo en la bbdd.

Podremos decidir cuando se ejecutan y que se ejecuta.

Estas operaciones pueden ser de actualización (UPDATE), inserción (INSERT) y borrado (DELETE).

# Un trigger es un procedimiento...

Por lo tanto, podrá modificar nuestra BBDD.

## Usos de los triggers

Nos permite registrar, auditar y monitorear los procesos de cambio de valores a las tablas de la base de datos activas.

Puede validar los valores aprobando o negando acciones realizadas por las sentencias SQL.

Puede preservar la consistencia y claridad de los valores, ejecutando acciones relacionadas con los campos de la bbdd.

- Recalcular el total de un pedido al añadir una línea, por ejemplo.

# Ventajas

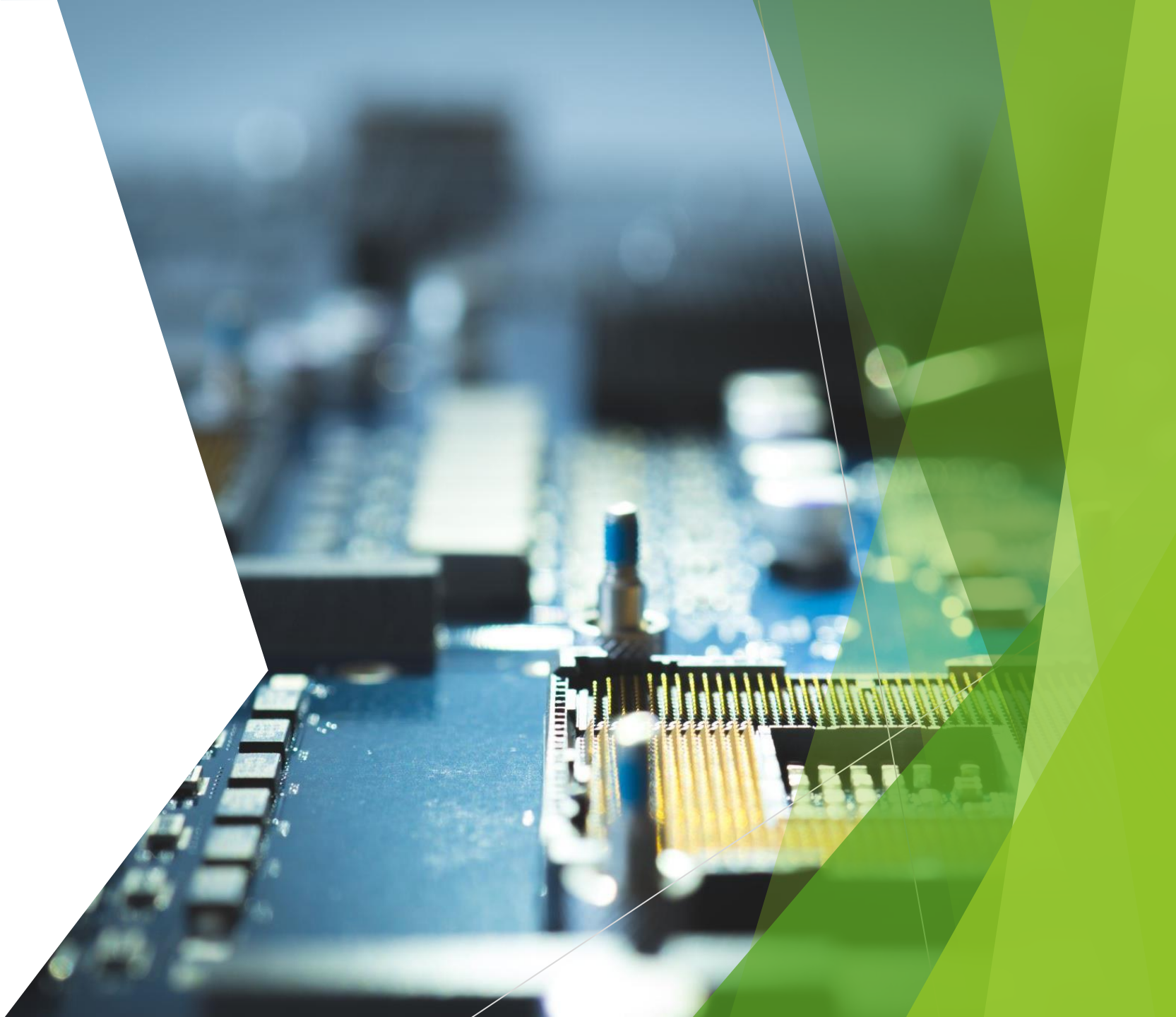
- ▶ Un trigger ofrece chequeos de seguridad en valores de las tablas de una base de datos.
- ▶ Fuerzan restricciones dinámicas de integridad de datos y de integridad referencial
- ▶ Aseguran que las operaciones relacionadas se realizan juntas de forma implícita

# Desventajas

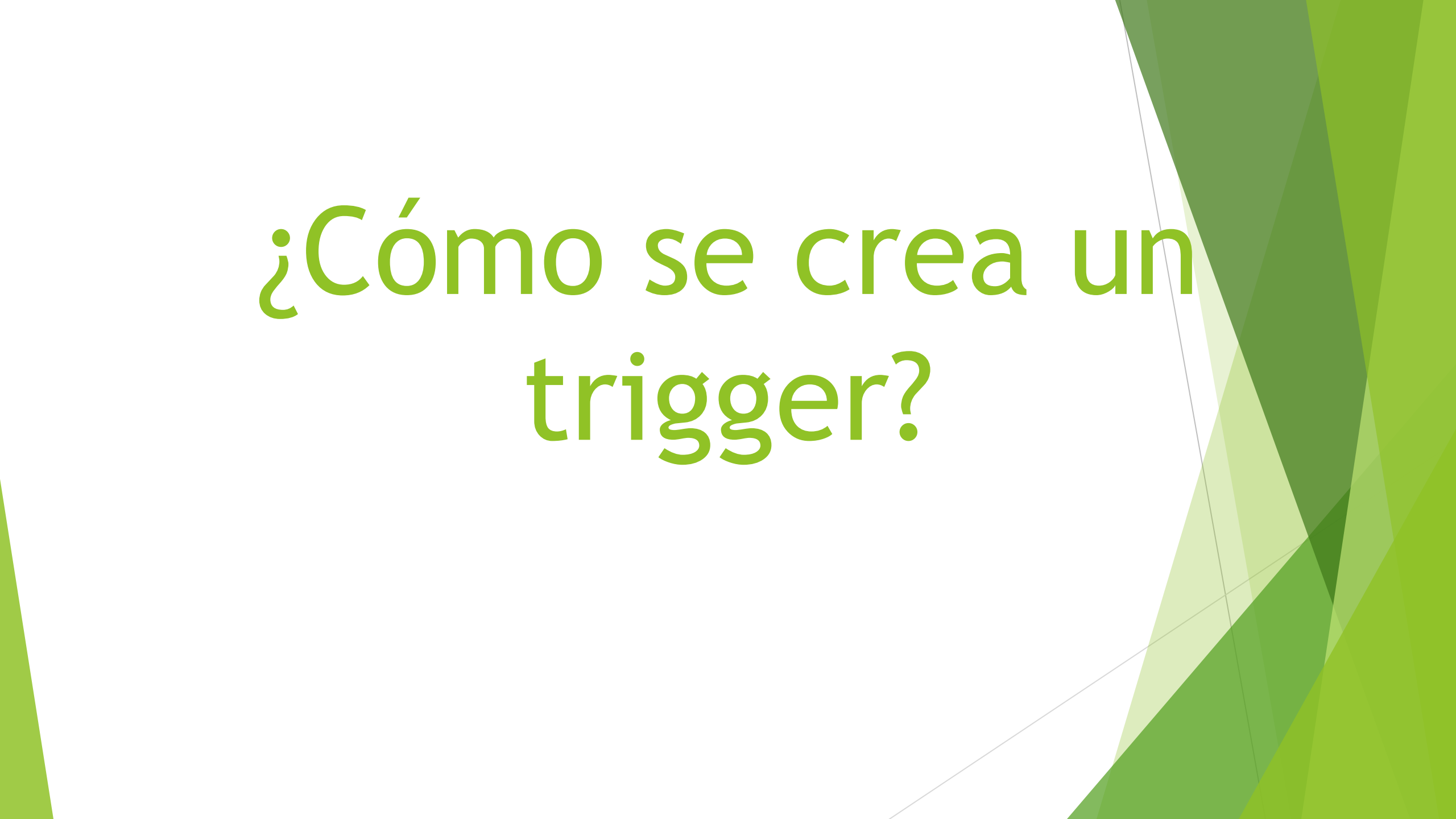
- ▶ Se tiene que programar anticipadamente lo que tiene que realizar un trigger
- ▶ Aunque es un procedimiento no se puede invocar directamente
- ▶ Los triggers siempre serán creados para un conjunto de registros y no para uno solo ya que se dispara por operación SQL

## 2 tipos de triggers:

- ▶ Sobre una tabla
- ▶ Sobre una BBDD o Servidor





The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

¿Cómo se crea un  
trigger?

## Sobre una tabla...

```
CREATE TRIGGER NOMBRE_TRIGGER  
ON [TABLE | VIEW ]  
FOR | INSTEAD OF  
[ INSERT ][ , ][ UPDATE ][ , ] [ DELETE ]  
AS  
SENTENCIA_SQL
```

# FOR o INSTAED OF

- ▶ FOR si queremos que se ejecute DESPUES del evento que ha disparado el trigger
- ▶ INSTEAD OF si queremos que se ejecute EN LUGAR del evento que ha disparado el trigger.

# Sobre una bbdd o servidor

```
CREATE TRIGGER NOMBRE_TRIGGER  
ON [ALL SERVER | DATABASE ]  
FOR  
AS  
SENTENCIA_SQL
```

- Queremos guardar en una tabla Menu\_HTO todo cambio de precio en la tabla menu con la fecha en que se ha cambiado.

Vamos a crear  
un triggers

Creamos una  
tabla...

```
CREATE TABLE Menu_HCO (  
    Id int IDENTITY(1,1) PRIMARY KEY,  
    IdMenu int NOT NULL,  
    Nombre Varchar(100) NOT NULL,  
    PrecioVenta varchar(100) NOT NULL,  
    Fecha DateTime NOT NULL  
)
```

# Creamos el Trigger

```
CREATE TRIGGER HistoricoPrecio
ON MENU
FOR UPDATE,INSERT
AS
BEGIN
    SET NOCOUNT ON;
    IF UPDATE(PrecioVenta)-- Solo si se actualiza pvp
    BEGIN
        INSERT INTO MENU_HCO
        SELECT id,nombre,precioVenta, getdate()
        FROM INSERTED
    END
END
```

# Comprobemos si funciona...

- ▶ Cambia el precio de la arepa albina a 4.00€
- ▶ Cambia el nombre de los tequeños a Pequeños
- ▶ Mira la tabla Menu\_hco



# Tablas Inserted y deleted



Nos servirán en nuestros triggers para recuperar los valores que se han modificado y han disparado nuestro trigger.



En un trigger de update, insert, tendremos la tabla inserted con los valores insertados.



En un trigger de delete, update, la tabla deleted con los valores borrados.

# Creamos un trigger sobre la BBDD

Vamos a crear un trigger que nos impida hacer un drop table.

```
CREATE TRIGGER SeguridadBorrarTabla  
ON DATABASE  
FOR DROP_TABLE  
AS  
PRINT 'Para borrar esta tabla debes deshabilitar el  
Trigger SeguridadBorrarTabla'  
ROLLBACK
```

Trigger  
sobre bbdd

Intenta  
borrar una  
tabla..

Drop table menu\_hto



# Podemos deshabilitar, habilitar o borrar un trigger

**DISABLE** TRIGGER HistoricoPrecio  
ON **Menu**

O...

**ENABLE** TRIGGER HistoricoPrecio  
ON Menu

**DISABLE** trigger SeguridadBorrarTabla on **DATABASE**

```
CREATE TRIGGER connection_limit_trigger
ON ALL SERVER
FOR LOGON
AS
BEGIN
    IF ORIGINAL_LOGIN()='sa' AND
        (SELECT COUNT(*) FROM sys.dm_exec_sessions
        WHERE is_user_process = 1 AND
        original_login_name = 'sa') > 3

        ROLLBACK;
END;
```

Trigger para  
impedir 3  
accesos  
simultaneos al  
usuario sa

# Eventos en los triggers sobre Servidor/Database

- ▶ Aquí teneis la lista de eventos disponibles...
- ▶ <https://docs.microsoft.com/es-es/sql/relational-databases/triggers/ddl-events?view=sql-server-ver15>



- ▶ La tabla pedidos y pedidoslinea del arepazo están relacionadas. Queremos que al añadir una línea a un pedido, se recalcule automáticamente el importe total de la tabla pedidos.
  - ▶ Analiza cómo funcionan los importes de esas tablas
  - ▶ Haz una consulta que dado un número de pedido, te sume el importe \* la cantidad de cada línea que haya en pedidoslinea
  - ▶ Escribe un update que actualice el campo total de la tabla pedidos y lo sobrescriba con la suma calculada anteriormente
  - ▶ Escribe un trigger, que cuando se inserte o modifique una línea, se autocalcule el importe total del pedido.

## Ejercicio



A top-down view of ingredients for arepas on a grey surface. There is a large white bowl of white flour, a metal whisk, a carton of eggs with several white eggs visible, a cracked eggshell, a small glass bowl of butter cubes, a small white bowl of yellow liquid (likely egg yolk), and a white pitcher of milk.

# Ejercicio

Quiero que creéis un trigger para que, cuando se añada un elemento al menú de la categoría Arepas (id cat = 1, se le añada el ingrediente masa arepa (id ingrediente=1) a la receta automáticamente.

# Ejercicio

Crear un trigger para que guarde en una tabla historico, todo cambio o inserción en la tabla clientes, guardando la fecha actual en el registro.

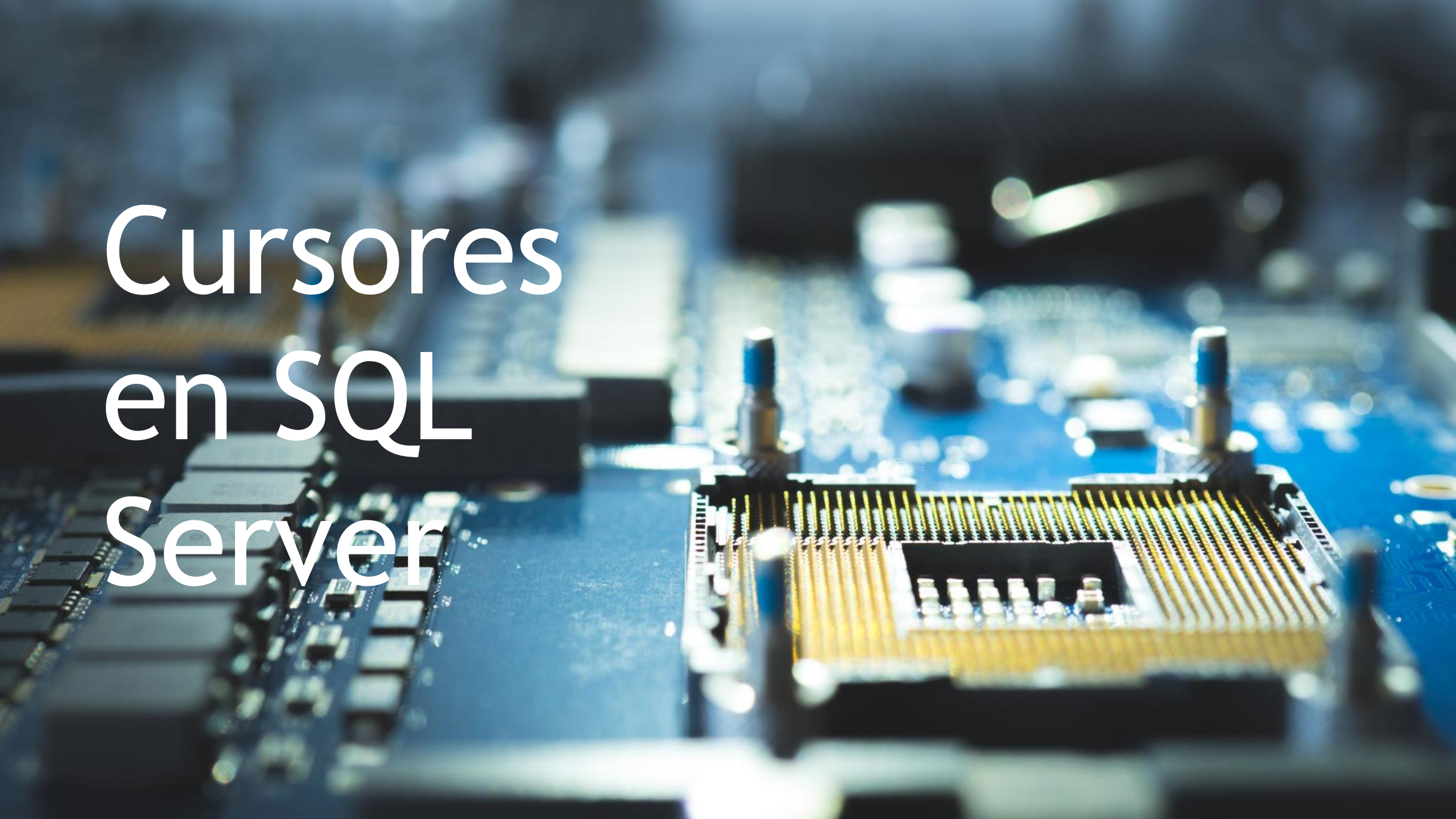
# Ejercicio

Crea un trigger que cuando alguien modifique el precio del menú, se guarde en la tabla menu\_htco (historico), el usuario de la bbdd que lo ha modificado, así como la hora.

# Ejercicio

- ▶ Crea un trigger para que cuando se modifique o inserte un valor en el pedido del arepazo calcule y modifique los gastos de envío según el código postal.
  - ▶ Si empieza por 410 -> gastos de envío 3.99
  - ▶ Si no, si empieza por 41 -> gastos de envío 5,99
  - ▶ Para el resto, 9,99.





# Cursores en SQL Server

# ¿Para que sirve un cursor?

- ▶ Nos servirán para recorrer fila a fila una tabla desde un procedimiento o función.
- ▶ El uso de cursores implicará 5 fases.
  - ▶ Declaración, Apertura, Acceso a datos, Cierre y Desalojo.

# Las 5 fases

## Declaración

- Deberemos declarar nuestro cursor igual que declaramos una variable.

## Apertura

- El cursor se tiene que abrir para poder ser usado.

## Acceso a datos

- Accederemos a los datos de la fila desde un bucle en TSQL

## Cierre

- Debe cerrarse el cursor antes de liberar la memoria.

## Desalojo.

- Liberamos la memoria del cursor y lo destruimos.



# Declaración

```
DECLARE CursorCliente CURSOR FOR SELECT email FROM Clientes
```



# Apertura

- ▶ En este momento, es donde se ejecuta la consulta y se guarda en nuestro cursor.
- ▶ El cursor se puede interpretar como un Array con los resultados de la consulta.

**OPEN** CursorCliente

# Acceso a datos

- ▶ Metemos el primer registro de nuestro array en una variable y movemos el cursor al siguiente registro.
- ▶ La variable @emailcliente hay que declararla antes.
  - ▶

```
FETCH NEXT FROM CursorCliente INTO @emailcliente
```

# ¿Y cuando dejo de leer?

- ▶ Como se cuando he llegado al final?
- ▶ Gracias a la variable @@FETCH\_STATUS que nos devolverá 0 cuando no haya mas filas.

...

```
WHILE @@fetch_status = 0
```

```
BEGIN
```

```
...
```

```
END
```

# Cierre y desalojo

- El cursor hay que cerrarlo fuera del bucle.

```
CLOSE CursorCliente
```

También debemos liberar la memoria del cursor con la instrucción:

```
DEALLOCATE CursorCliente
```

# A ver todo junto...

```
DECLARE @emailCliente AS nvarchar(400)
DECLARE CursorEmail CURSOR FOR SELECT email FROM Clientes
OPEN CursorEmail
FETCH NEXT FROM CursorEmail INTO @emailCliente
WHILE @@fetch_status = 0
    BEGIN
        PRINT 'El email es: ' + @emailCliente
        FETCH NEXT FROM CursorEmail INTO @emailCliente
    END
CLOSE CursorEmail
DEALLOCATE CursorEmail
```

# Y si queremos recuperar + de un campo?

```
DECLARE @emailCliente AS nvarchar(400)
DECLARE @nombre AS nvarchar(400)
DECLARE CursorEmail CURSOR FOR SELECT email,nombre FROM Clientes
OPEN CursorEmail
FETCH NEXT FROM CursorEmail INTO @emailCliente,@nombre
WHILE @@fetch_status = 0
    BEGIN
        PRINT 'El email es: ' + @emailCliente + ' y el nombre es:' + @nombre
        FETCH NEXT FROM CursorEmail INTO @emailCliente,@nombre
    END
CLOSE CursorEmail
DEALLOCATE CursorEmail
```

# Ejercicio

Crea una función que devuelva una cadena con todos los emails de la tabla clientes separados por ,

# Ejercicio

Vamos a complicar el ejercicio anterior. Quiero que devuelva una tupla por cada email encontrado en la tabla clientes. Una tupla formada por el email y el numero de pedidos que ha realizado el cliente separados por |.

Tiene que devolver algo asi:

Pepe@gmail.com|23, angeles@yahoo.es|5, ...