BD2

CONCEPTOS CLAVE EN LA OPCIÓN PROCEDIMENTAL

SERGIO ÁLVAREZ

VERSIÓN 2.3

SINTAXIS PASCAL

DECLARACIÓN Y ASIGNACIONES

```
DECLARE var1, var2 INT DEFAULT 0;
DECLARE str1 VARCHAR(50);
DECLARE today DATE DEFAULT CURRENT_DATE;
DECLARE v1, v2, v3  DOUBLE(10,2);

set var1 = 101;
set str1 = 'Hello world';
set v1 = 19.99;
```

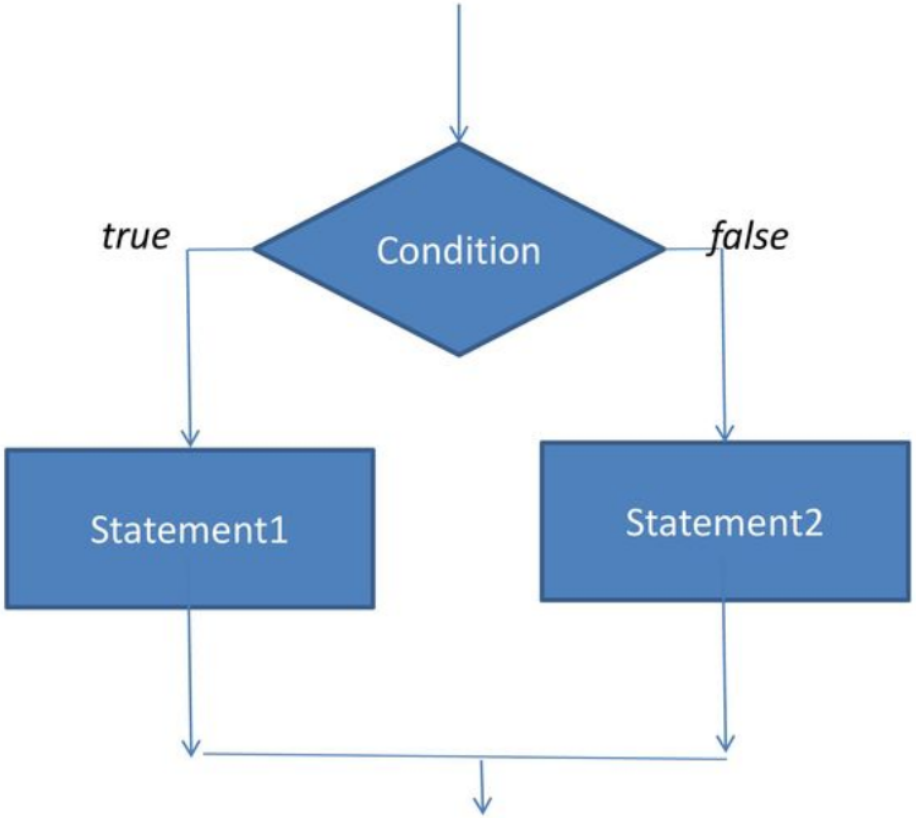
Reglas:

1. La declaración de las variables SIEMPRE ESTAN DESPUES DEL BEGIN.
2. Después de una instrucción NO PUEDE DECLARA MAS VARIABLES.
3. Hay tres tipos de variables y se deben poner en este orden:
    1. Variables normales (int, varchar, double, etc)
    2. Variables cursores
    3. Variables excepciones

CONDICIONAL

Conditional Operator

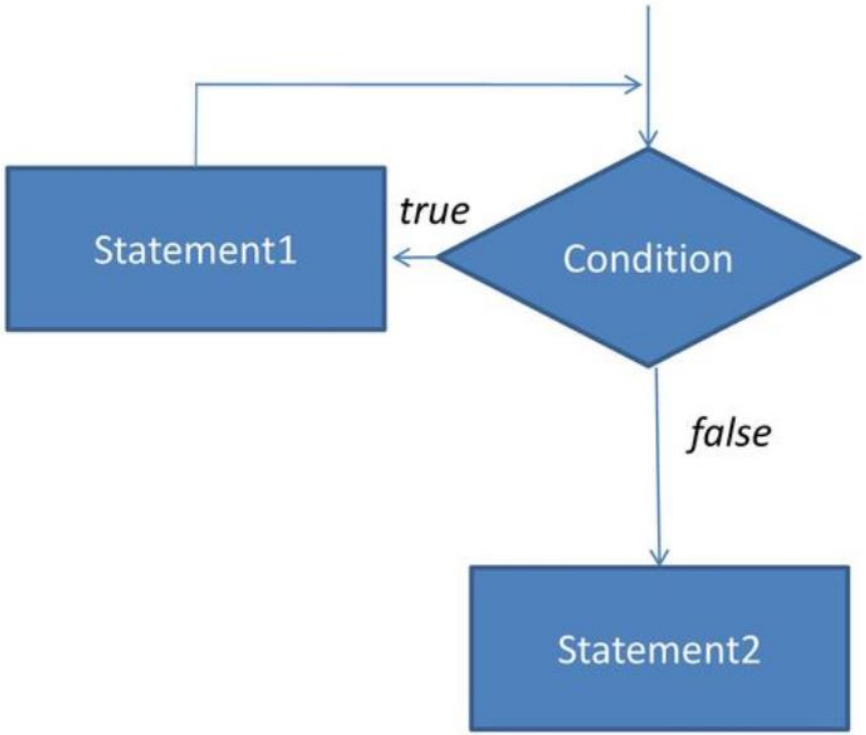
```
if condition then
  statement1;
else
  statement2;
End if;
```



CICLO REPITA MIENTRAS QUE

While Loop

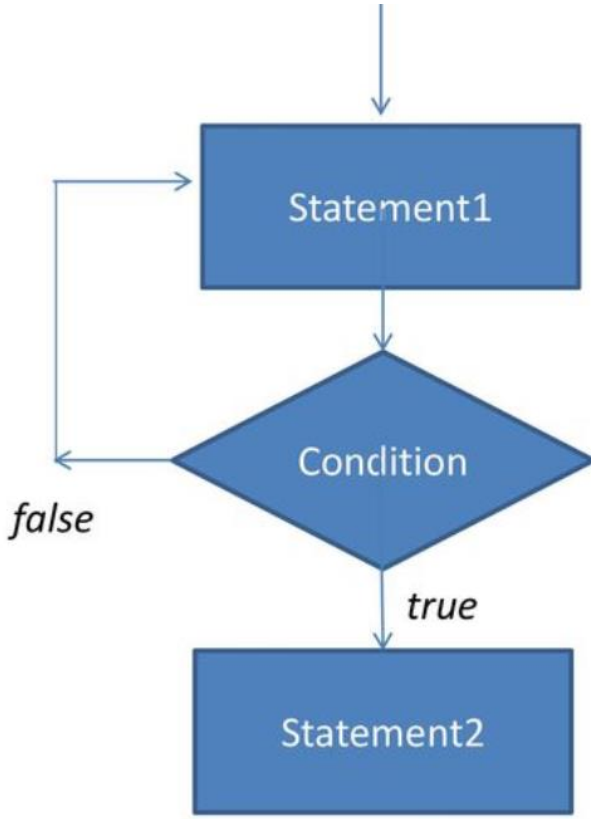
```
declare var1 int default 1;

myloop1:
    while (var1 <=10) do
        select var1;
        if  var1 > 3 then
            leave myloop1;
        end if;
        set var1 = var1 + 1;
    end while;
    select 'adios';
```



CICLO REPITA HASTA

Repeat Loop

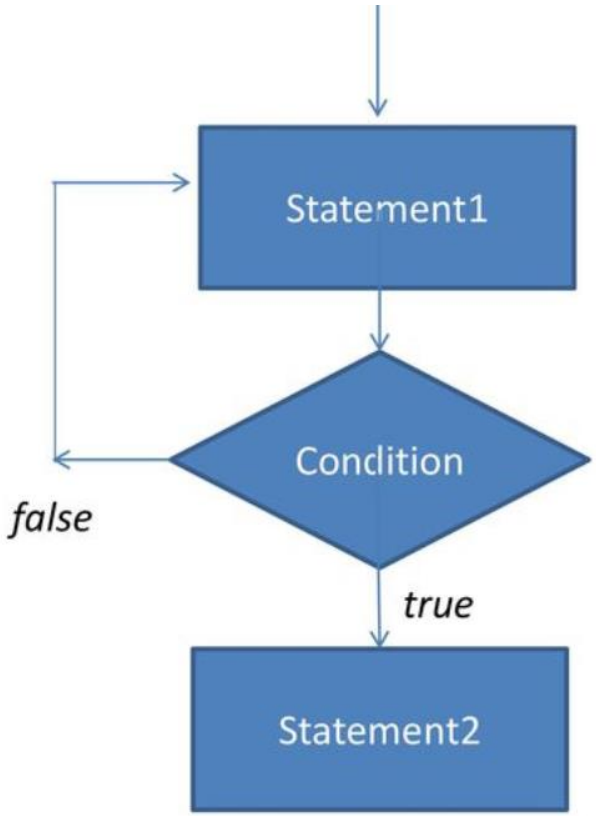
```
declare var1 int default 1;

myloop1: repeat
        select var1;
        if  var1 > 3 then
                leave myloop1;
        end if;
        set var1 = var1 + 1;
   until var1 > 5 end repeat;
   select 'adios';
```



CICLO

Loop

```
declare var1 int default 1;

myloop1: loop
        select var1;
        if  var1 > 3 then
                leave myloop1;
        end if;
        set var1 = var1 + 1;
   end   loop  myloop1;
   select 'adios';
```



FUNCIONES Y PROCEDIMIENTOS

```
DELIMITER $$

CREATE FUNCTION `function-name` ( parameter TYPE )
RETURNS output-type

BEGIN
        statements;
        return some_value;
END $$
```

```
DELIMITER $$

CREATE PROCEDURE `procedure_name`( [[IN | OUT | INOUT] parameter1 TYPE] )
BEGIN
        statements;

END $$
```

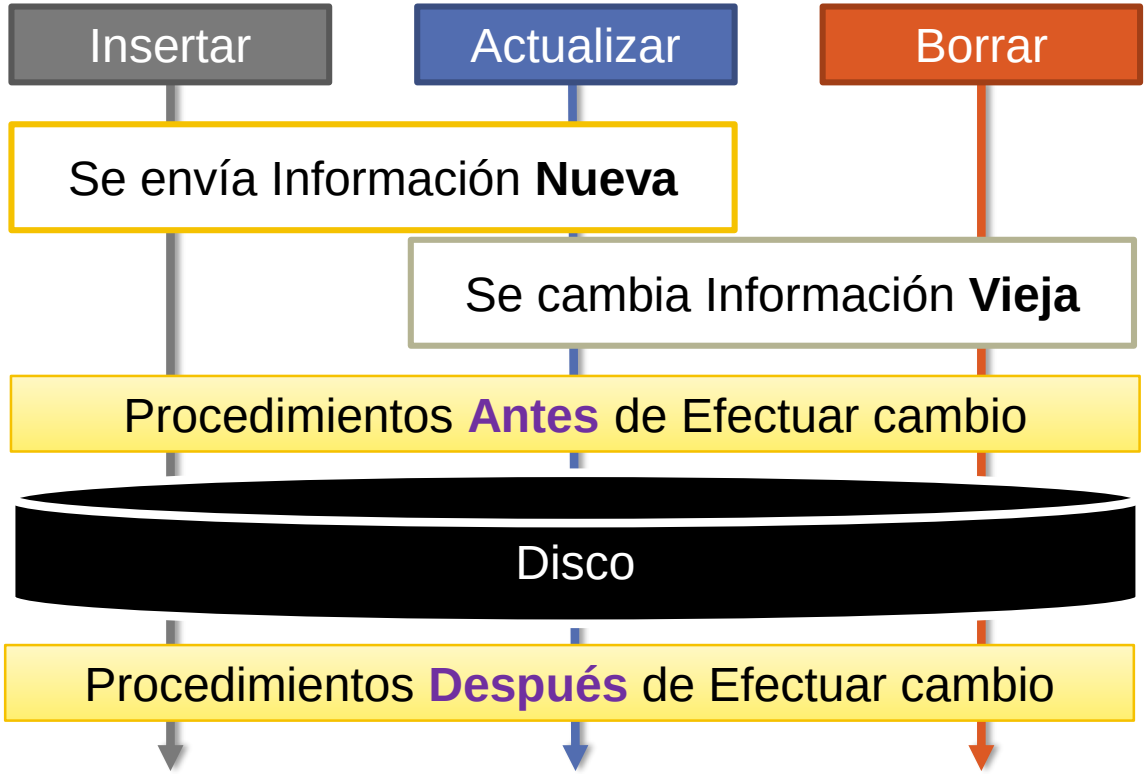
TRIGGERS

TRIGGERS

Insert into **usuarios**
(id, Nombre, Apellido)
Values('4387978',**'Ana**','Zapata');

Update **usuarios**
set Nombre='**Mariana**'
where id = '4387978';

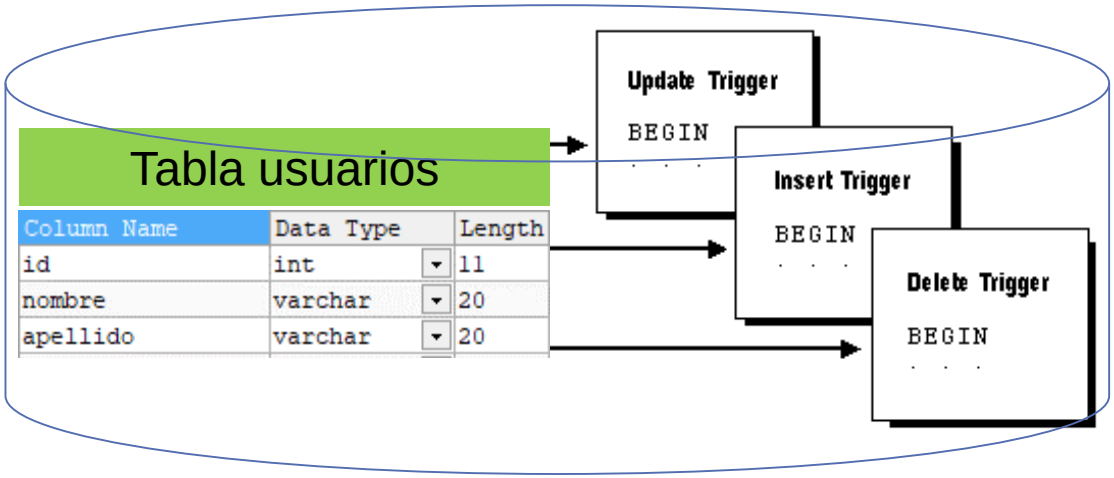
Delete from **usuarios**
where id = '4387978';



TRIGGERS ESTRUCTURAS OLD Y NEW

DML

```
update usuarios set ...;

insert into usuarios ...;

delete from usuarios ...;
```



Como se pasar “argumentos” en otras palabras como puedo saber que es nuevo y que es viejo en la base de datos?

Datos nuevos

Datos Ya en la Base de datos

DML	Insert	Update	Delete	Ejemplo
NEW	SI	SI	NO	new.id new.nombre new.apellido
OLD	NO	SI	SI	old.id old.nombre old.apellido

TRIGGERS EJEMPLO NEW Y OLD

Tiempo

Insert into **usuarios**
(id, Nombre, Apellido)
Values('4387978',**'Ana**','Zapata');

Update usuarios
set Nombre='**Mariana**'
where id = '4387978';

Delete from **usuarios**
where id = '4387978';

Primera acción Insert	Segunda acción Update	Tercera acción Delete

`new.id = '4387978'`
`new.nombre = 'Ana'`
`new.apellido = 'Zapata'`
	`new.id = '4387978'`
	`new.nombre = 'Mariana'`
	`new.apellido = 'Zapata'`
		X

X	`old.id = '4387978'`
	`old.nombre = 'Ana'`
	`old.apellido = 'Zapata'`
		`old.id = '4387978'`
		`old.nombre = 'Mariana'`
		`old.apellido = 'Zapata'`

SINTAXIS TRIGGER

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER  trigger_name
            [ BEFORE |  AFTER ]
            trigger_event  ON  tbl_name
            FOR EACH ROW  trigger_body
BEGIN
   Body-of-the-trigger;
END;
```

trigger_event

INSERT
DELETE
UPDATE

HACER UN LOG DE CAMBIOS

Tabla usuarios

Column Name	Data Type	Length
id	int	11
nombre	varchar	20
apellido	varchar	20

Crear el trigger

```
create trigger bu_usuarios
  before update on usuarios
  for each row
begin
 insert into mi_log_usuarios values(
                concat( OLD.nombre,',',OLD.apellido),
                concat( NEW.nombre,',',NEW.apellido));
end;
```

Ejecutar el trigger

```
update usuarios
  set nombre = 'Juan',
      apellido = 'Zapata'
  where id = 1;
```

TRIGGERS MODIFICAR COLUMNA

usuarios

Column Name	Data Type	Length
id	int	11
nombre	varchar	20
apellido	varchar	20
nombre completo	varchar	41

Se quiere que cuando se inserte un registro en la tabla de usuarios el nombre completo sea la concatenación del nombre y el apellido

Crear el trigger

```
DELIMITER $$
CREATE TRIGGER antes_insertar_en_usuarios BEFORE INSERT ON usuarios
FOR EACH ROW BEGIN
        /*Se Modifica la columna full_name*/
        SET NEW.nombre_completo = CONCAT(NEW.nombre, ' ', NEW.apellido);
        /*sigue con el proceso de insertar el registro*/
END$$
DELIMITER ;
```

Ejecutar el trigger

```
INSERT INTO usuarios ( nombre,apellido) VALUES ('Sergio','Alvarez');
```

id	nombre	apellido	nombre_completo
1	Sergio	Alvarez	Sergio Alvarez

TRIGGER EXCEPCIONES

La instrucción SIGNAL detiene el proceso de inserción, actualización o borre.

El SQLState 45000 es “excepción definida por el usuario”

```
SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Problem – blah... ‘;
```

TRIGGERS

Crear el trigger

```
DROP TRIGGER antes_insertar_en_usuarios;
DELIMITER $$
CREATE TRIGGER antes_insertar_en_usuarios BEFORE INSERT ON usuarios
FOR EACH ROW BEGIN
  /*Validación*/
  IF NEW.nombre IS NULL OR LENGTH( NEW.nombre ) < 3 THEN
    /*Si no cumple se detiene la ejecución (NO INSERTA)*/
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'El nombre no puede ser nulo y longitud menor a 3';
  END IF;
  /*Se Modifica la columna full_name*/
  SET NEW.nombre_completo = CONCAT(NEW.nombre, ' ', NEW.apellido);
  /*sigue con el proceso de insertar el registro*/
END$$
DELIMITER ;
```

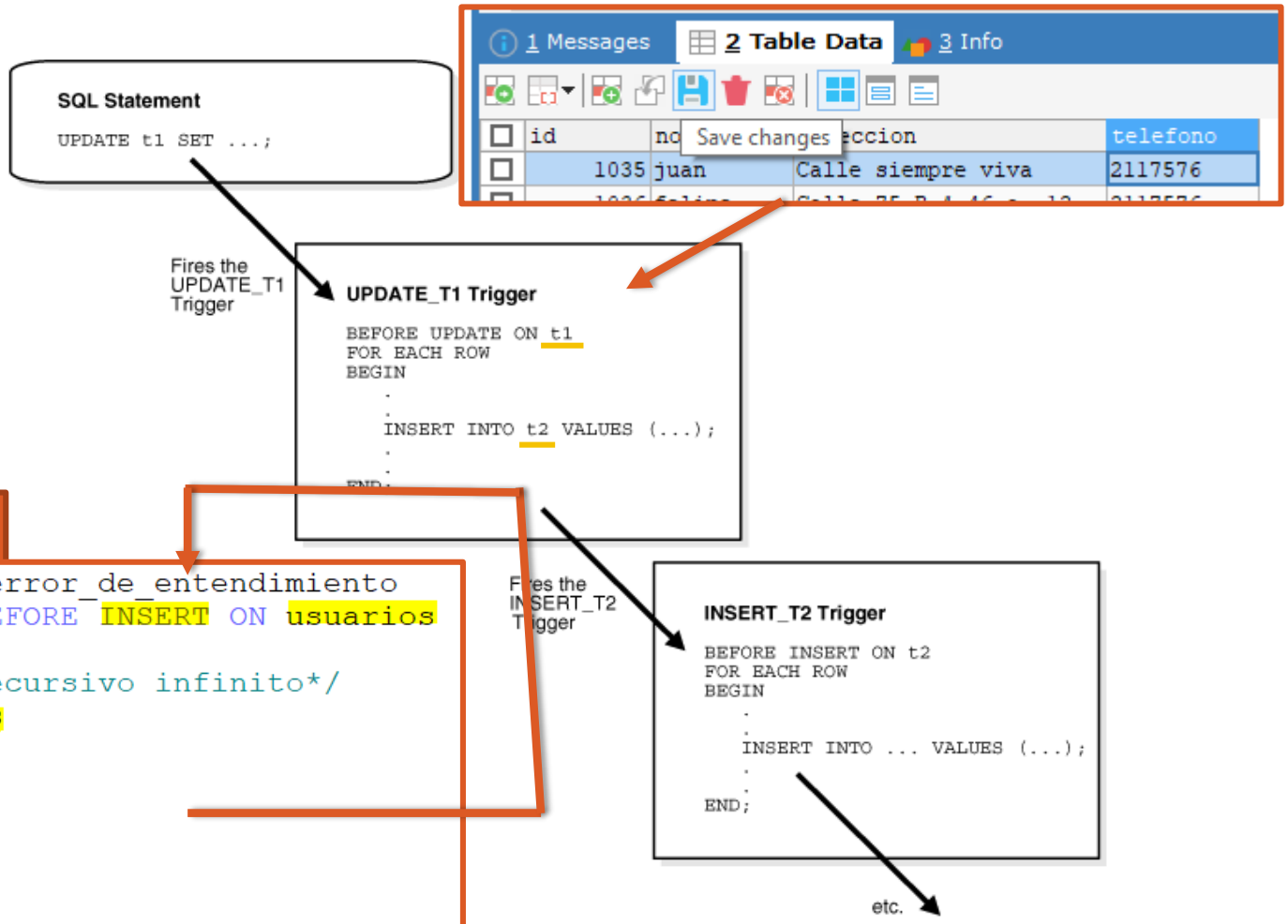
Ejecutar el trigger

```
1   INSERT INTO usuarios ( nombre,apellido) VALUES ('Lu','Gomez');
```

1 Messages | 2 Table Data | 3 Info

```
1 queries executed, 0 success, 1 errors, 0 warnings

Query: INSERT INTO usuarios ( nombre,apellido) VALUES ('Lu','Gomez')

Error Code: 1644
El nombre no puede ser nulo y longitud menor a 3
```

ERROR TRIGGERS RECURSIVOS!!



NO HACER ESTO!!!

```
CREATE TRIGGER super_error_de_entendimiento
                    BEFORE INSERT ON usuarios
FOR EACH ROW BEGIN
  /*ERROR!! Trigger recursivo infinito*/
  INSERT INTO usuarios
    ( nombre,
      apellido,
      nombre_completo)
  VALUES (
      NEW.nombre,
      NEW.apellido,
      CONCAT(NEW.nombre, ' ', NEW.apellido));
END;
$$
```

CURSORES

QUE SON LOS CURSORES?

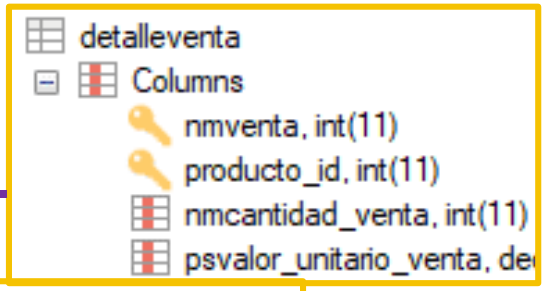
Un CURSOR es un tipo de apuntador creado en la base de datos que provee acceso secuencial (una fila al tiempo) a las sentencias SELECT.

Hay dos tipos de cursores

- **Implícitos** – Tienen que recuperar una y solo una fila
- **Explícitos** – Pueden recuperar cualquier cantidad de filas

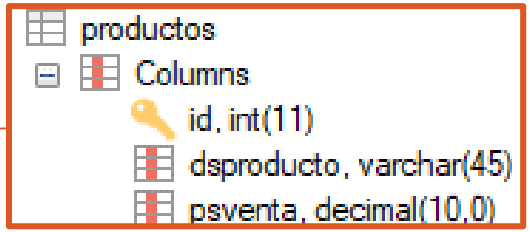
Wikipedia: Un cursor se utiliza para el procesamiento individual de las filas devueltas por el sistema gestor de base de datos para una consulta. Es necesario debido a que muchos lenguajes de programación sufren de lo que en inglés se conoce como **impedance mismatch**. Por norma general los lenguajes de programación son procedurales y no disponen de ningún mecanismo para manipular conjuntos de datos en una sola instrucción. Debido a ello, las filas deben ser procesadas de forma secuencial por la aplicación. Un cursor puede verse como un iterador sobre la colección de filas que habrá en el set de resultados.

CURSORES IMPLICITOS



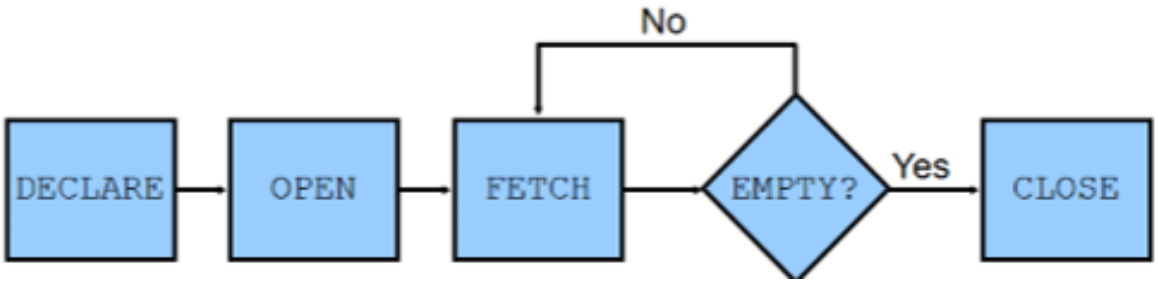
NEW.

```
CREATE TRIGGER detalleventa_bi BEFORE INSERT ON detalleventa
FOR EACH ROW BEGIN
  /*Variable normal*/
  DECLARE v_psventa DECIMAL(10,0);
  /*Seleccionar una fila de otra tabla*/
  SELECT psventa
    INTO v_psventa
    FROM productos
    WHERE id = new.producto_id;
  /*Hacer validación*/
  IF v_psventa > new.psvalor_unitario_venta THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Valor venta menor que el de compra';
  END IF;
END$$
```



Usados para recuperar múltiples **columnas** de **un solo registro**.

CURSORES EXPLICITOS



Declare: Define la consulta a realizar (es decir el Select), es similar a una variable

```
DECLARE cursor_name CURSOR FOR SELECT Columna1,Columna2... FROM...
```

Open: Ejecuta el Select. En este momento se recupera la información de la base de datos (Se podría ver como si se creara una tabla temporal)

```
OPEN cursor_name
```

Fecth: “Intenta” leer una fila. Se debe intentar primero para validar si hay información, esto conlleva a que se debe validar si fallo el intento.

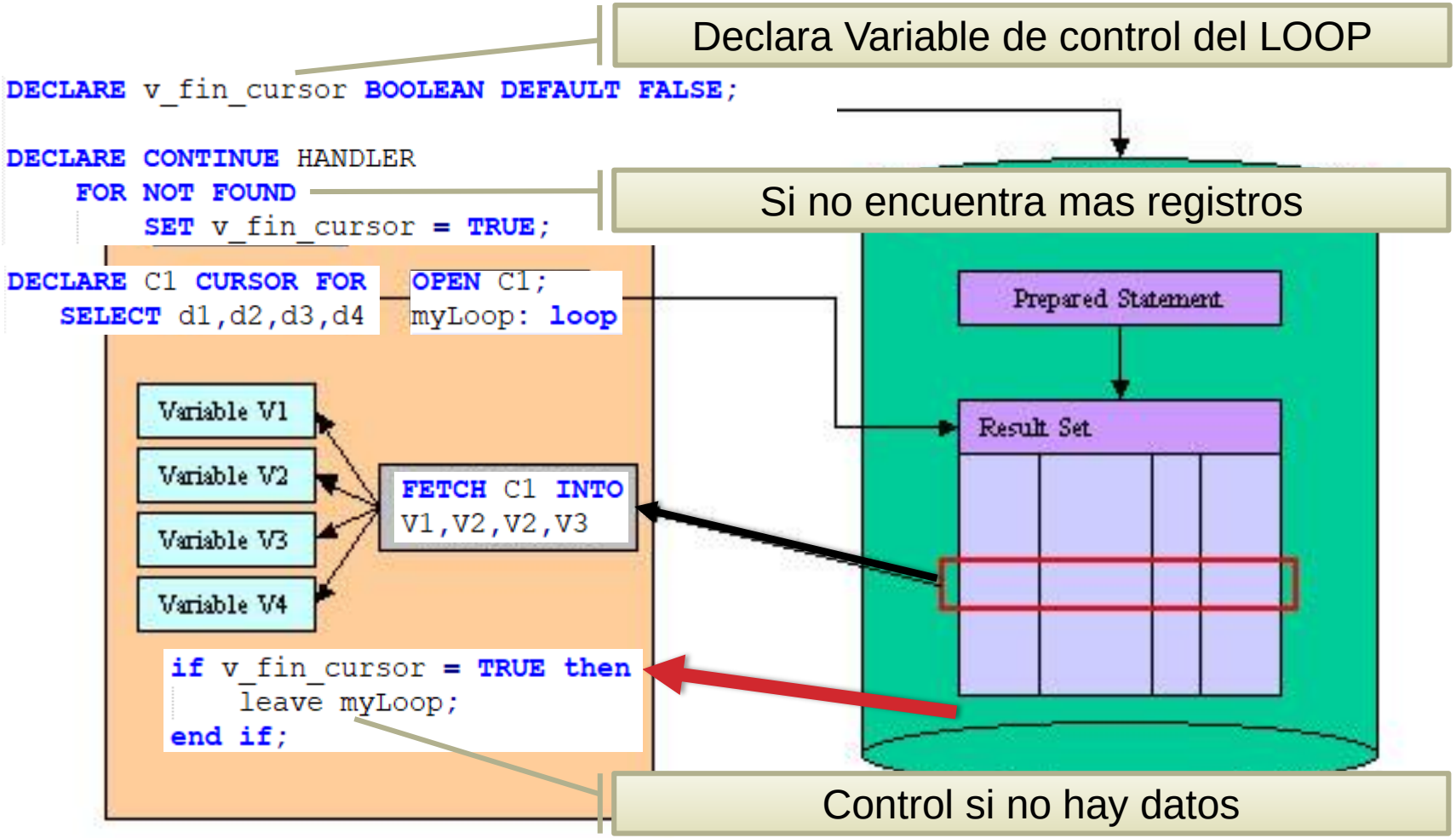
```
FETCH cursor_name INTO variable1, variable2...
```

Close: Libera la memoria

```
CLOSE cursor_name
```

CURSORES EXPLÍCITOS

Ejemplo leer registro por registro, realizando una operación/validación con variable de control v_fin_cursor

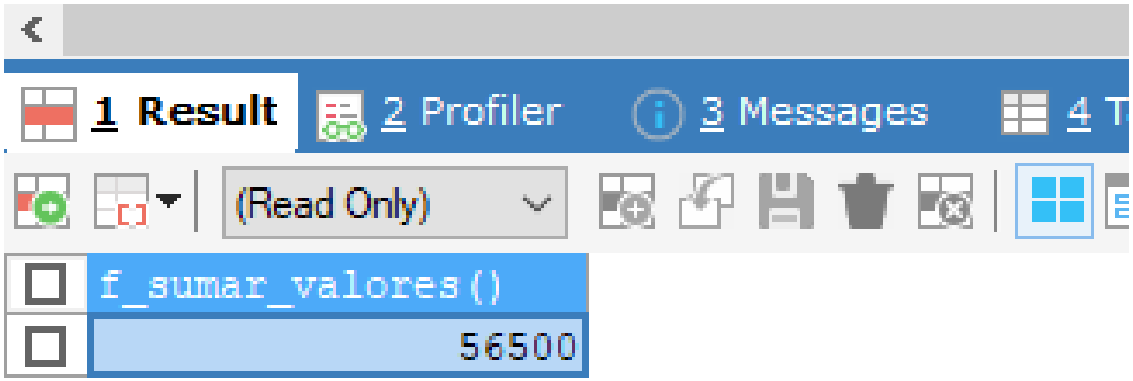


VARIABLES Y CURSORES

```
CREATE FUNCTION f_sumar_valores()    RETURNS INT
BEGIN
/*Variables Normales*/
DECLARE v_fin_cursor INT DEFAULT FALSE;
DECLARE v_psventa INT;
DECLARE v_suma INT DEFAULT 0;
/*Cursores */
DECLARE C_productos CURSOR FOR
        SELECT psventa
          FROM productos;
/*Manejo de excepciones*/
DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_fin_cursor = TRUE;

/*Inicio Codigo Normal*/
OPEN C_productos;/*Se ejecuta la consulta del cursor*/
FETCH C_productos INTO v_psventa; /*intenta leer primer registro*/

  /*ciclo*/
  loopCursor: WHILE v_fin_cursor = FALSE DO
        SET v_suma = v_suma + v_psventa;
        FETCH C_productos INTO v_psventa; /*intenta leer siguientes*/
  END WHILE;

CLOSE C_productos;
RETURN v_suma;
END$$
```

EJECUTAR FUNCIÓN

```
1    SELECT f_sumar_valores();
2
```



PREGUNTAS