Práctica de Triggers

1. Dada la tabla stock de la base de datos stores7 se requiere crear una tabla stock_historia_precios que almacene los cambios de precios que haya habido.

   Tabla stock_historia_precios

   - Stock_historia_Id INT Identity
   - Stock_num
   - Manu_code
   - fechaYhora (grabar fecha y hora del evento)
   - usuario (grabar usuario que realiza el cambio de precios)
   - unit_price_old
   - unit_price_new
   - estado char default 'A' check (estado IN ('A','I'))

```
CREATE TABLE stock_historia_precios (
        stock_historia_id int IDENTITY(1,1) PRIMARY KEY,
        stock_num smallint,
        manu_code char(3),
        fechaYHora datetime,
        usuario varchar(20),
        unit_price_old decimal(6,2),
        unit_price_new decimal(6,2),
        estado char DEFAULT 'A' CHECK(estado IN('A','I')),
);
GO

CREATE TRIGGER cambio_precios_stock ON stock
AFTER UPDATE
AS
BEGIN
        DECLARE @unit_price_old decimal(6,2)
        DECLARE @unit_price_new decimal(6,2)
        DECLARE @stock_num smallint
        DECLARE @manu_code char(3)

        DECLARE precios_stock CURSOR FOR
        SELECT i.stock_num,i.manu_code, i.unit_price, d.unit_price FROM inserted i JOIN
deleted d ON (i.stock_num = d.stock_num)
        WHERE i.unit_price != d.unit_price

        OPEN precios_stock

        FETCH NEXT FROM precios_stock
        INTO @stock_num, @manu_code, @unit_price_new, @unit_price_old

        WHILE @@FETCH_STATUS = 0
        BEGIN
```

```
			INSERT INTO stock_historia_precios(stock_num, manu_code, unit_price_new, 
unit_price_old, fechaYHora, usuario)
			VALUES(@stock_num, @manu_code, @unit_price_new, @unit_price_old, 
GETDATE(), CURRENT_USER)

		FETCH NEXT FROM precios_stock
		INTO @stock_num, @manu_code, @unit_price_new, @unit_price_old

		END

		CLOSE precios_stock
		DEALLOCATE precios_stock
END;
```

2. Crear un trigger sobre la tabla stock_historia_precios que ante un delete sobre la misma realice en su lugar un update de campo estado de ‘A’ a ‘I’ (inactivo).

```
CREATE TRIGGER delete_stock_historia ON stock_historia_precios
INSTEAD OF DELETE
AS
BEGIN
		DECLARE @stock_historia_id int

		DECLARE stock_historia_borrado CURSOR FOR
		SELECT stock_historia_id FROM deleted

		OPEN stock_historia_borrado

		FETCH NEXT FROM stock_historia_borrado
		INTO @stock_historia_id

		WHILE @@FETCH_STATUS = 0
		BEGIN
			UPDATE stock_historia_precios SET estado = 'I' WHERE stock_historia_id = 
@stock_historia_id

		FETCH NEXT FROM stock_historia_borrado
		INTO @stock_historia_id

		END

		CLOSE stock_historia_borrado
		DEALLOCATE stock_historia_borrado

END;
GO
```

3. Validar que sólo se puedan hacer inserts en la tabla stock en un horario entre las 8:00 AM y 8:00 PM. En caso contrario enviar un error por pantalla.

```
CREATE TRIGGER inserts_stock ON stock
INSTEAD OF INSERT
AS
BEGIN
	IF(DATEPART(HOUR, GETDATE()) BETWEEN 8 AND 20)
	BEGIN
		INSERT INTO stock(stock_num, manu_code, description, unit, unit_descr, 
unit_price)
		SELECT stock_num, manu_code, description, unit, unit_descr, unit_price FROM 
inserted
	END
	ELSE
	BEGIN
		RAISERROR('Maestro que haces a esta hora laburando', 12, 1)
	END

END;
GO
```

4. Crear un trigger que realice un borrado en cascada sobre las tablas orders e ítems, validando que sólo se borre 1 órden de compra.
   Si detecta que están queriendo borrar más de una orden de compra, informará un error y abortará la operación.

```
CREATE TRIGGER delete_orders_and_items ON orders
INSTEAD OF DELETE
AS
BEGIN
	DECLARE @customer_num smallint
	DECLARE @order_num smallint

	IF((SELECT COUNT(*) FROM deleted) > 1)
	BEGIN
		RAISERROR('No se pueden eliminar mas de una orden a la vez', 12, 1)
	END
	ELSE
	BEGIN
		SELECT @order_num = order_num, @customer_num = customer_num FROM 
deleted

		DELETE FROM items WHERE order_num = @order_num
		DELETE FROM orders WHERE order_num = @order_num AND customer_num 
= @customer_num
	END

END;
GO
```

5. Crear un trigger de insert sobre la tabla ítems que al detectar que el código de fabricante (manu_code) del producto a comprar, no existe en la tabla manufact, inserte una fila en dicha tabla con él manu_code ingresado, en el campo manu_name

la descipción ‘Fabricante Nro. de Orden 9999’ donde 9999 corresponde al nro. de la órden de compra a la que pertenece el ítem y en el campo lead_time el valor 1.

```
CREATE TRIGGER insert_items ON items
INSTEAD OF INSERT
AS
BEGIN
	DECLARE @manu_code char(3)
	DECLARE @order_num smallint

	DECLARE items_insertados CURSOR FOR
	SELECT manu_code, order_num FROM inserted

	OPEN items_insertados

	FETCH NEXT FROM items_insertados
	INTO @manu_code, @order_num

	WHILE @@FETCH_STATUS = 0
	BEGIN
		IF NOT EXISTS (SELECT * FROM manufact WHERE manu_code = @manu_code)
		BEGIN
			INSERT INTO manufact(manu_code, manu_name, lead_time)
			VALUES(@manu_code, 'Fabricante Nro. de orden ' +@order_num, 1)
		END

		FETCH NEXT FROM items_insertados
		INTO @manu_code, @order_num

	END

	CLOSE items_insertados
	DEALLOCATE items_insertados

	INSERT INTO items(item_num, order_num, manu_code, stock_num, quantity,
total_price)
	SELECT item_num, order_num, manu_code, stock_num, quantity, total_price FROM
inserted

END;
GO
```

6. Crear tres triggers (Insert, Update y Delete) sobre la tabla stock para replicar todas las operaciones en la tabla stock_replica, la misma deberá tener la misma estructura de la tabla stock.

```
CREATE TABLE stock_replica(
	stock_num smallint,
	manu_code char(3),
	description varchar(15),
	unit_price decimal(6,2),
	unit char(4),
	unit_descr varchar(15)
);
GO

alter table DBAS.stock_replica add constraint pk_stock_replica
	primary key clustered (stock_num, manu_code);
GO

CREATE TRIGGER replica_insert ON stock
AFTER INSERT
AS
BEGIN
	INSERT INTO stock_replica(stock_num, manu_code, description, unit, unit_descr,
unit_price)
	SELECT stock_num, manu_code, description, unit, unit_descr, unit_price FROM
inserted
END;
GO

CREATE TRIGGER replica_delete ON stock
AFTER DELETE
AS
BEGIN
	DELETE sr FROM stock_replica sr
	JOIN deleted d ON (sr.stock_num = d.stock_num AND sr.manu_code = d.manu_code)
END;
GO

CREATE TRIGGER replica_update ON stock
AFTER UPDATE
AS
BEGIN
	UPDATE sr SET sr.description = i.description, sr.unit = i.unit, sr.unit_descr =
i.unit_descr, sr.unit_price = i.unit_price
	FROM stock_replica sr
	JOIN inserted i ON (sr.stock_num = i.stock_num AND sr.manu_code = i.manu_code)
END;
GO
```

7. Crear la vista Productos_por_fabricante que tenga los siguientes atributos:

   Stock_num, manu_code, description, manu_name

   Crear un trigger de Insert sobre la vista anterior que ante un insert en la vista, en su lugar inserte una fila en la tabla stock, pero que valide que si el manu_code no existe en la tabla manufact, inserte además una fila en dicha tabla con el campo lead_time en 1.

```
CREATE VIEW productos_por_fabricante AS
SELECT s.stock_num, s.manu_code, s.description, m.manu_name FROM stock s JOIN manufact
m ON(s.manu_code = m.manu_code)
```

```sql
GO

CREATE TRIGGER insert_productos_por_fabricante ON productos_por_fabricante
INSTEAD OF INSERT
AS
BEGIN

	DECLARE @stock_num smallint
	DECLARE @manu_code char(3)
	DECLARE @description varchar(15)
	DECLARE @manu_name varchar(15)

	DECLARE insert_cursor CURSOR FOR
	SELECT stock_num, manu_code, description, manu_name FROM inserted

	OPEN insert_cursor

	FETCH NEXT FROM insert_cursor
	INTO @stock_num, @manu_code, @description, @manu_name

	WHILE @@FETCH_STATUS = 0
	BEGIN
		IF NOT EXISTS (SELECT * FROM manufact WHERE manu_code = @manu_code)
		BEGIN
			INSERT INTO manufact(manu_code, manu_name, lead_time)
			VALUES(@manu_code, @manu_name, 1)
		END

		INSERT INTO stock(stock_num, manu_code, description)
		VALUES(@stock_num, @manu_code, @description)

		FETCH NEXT FROM insert_cursor
		INTO @stock_num, @manu_code, @description, @manu_name

	END

	CLOSE insert_cursor
	DEALLOCATE insert_cursor

END;
GO
```