U.T. X.- CREACIÓN DE OBJETOS EN SQL: FUNCIONES Y DESENCADENADORES.

1.- FUNCIONES DEFINIDAS POR EL USUARIO.

- Una función es un módulo de programa que acepta 0, 1 ó más parámetros de entrada y devuelve un valor de salida.
  - ✓ La salida puede ser un único valor ó un conjunto de ellos en forma de tabla.
- Una función definida por el usuario se ejecuta igual que una función predeterminada, es decir, insertándola en el lugar apropiado de cualquier sentencia SQL.
  - ✓ El lugar dependerá del tipo de valor que devuelvan.
  - ✓ Para poder ejecutarse, el nombre de la función debe ir precedido del nombre del propietario (**dbo.)**

1.1.- CREACIÓN DE FUNCIONES.

- **Sintaxis:**

```
CREATE FUNCTION nombre_función
   ({ @nombre_parámetro Tipo_datos [ = Valor_inicial]} [, …] )
   RETURNS Tipo_datos
   [AS]
   BEGIN
   < instrucción SQL > …
   RETURN expresión
   END
```

1.2.- MODIFICACIÓN Y BORRADO DE FUNCIONES.

- **Sintaxis de Modificación de Funciones:**

```
ALTER FUNCTION nombre_función
   ({ @nombre_parámetro Tipo_datos [ = Valor_inicial]} [, …] )
   RETURNS Tipo_datos
   [AS]
   BEGIN
   < instrucción SQL > …
   RETURN expresión
   END
```

- **Sintaxis de Borrado de Funciones:**

```
DROP FUNCTION nombre_función
```

2.- TIPOS DE FUNCIONES: ESCALARES

- Las **funciones Escalares** devuelven un valor único.
  - ✓ Son similares a las funciones integradas.
- Pueden utilizarse en cualquier lugar de una sentencia SQL que admita un valor del tipo que la función devuelva.

2.- TIPOS DE FUNCIONES: EN LÍNEA

- Una **función en Línea** devuelve una tabla.
  - ✓ Puede entenderse como una vista con parámetros de entrada.
- Puede hacerse referencia a una **función en Línea** en cualquier lugar de una sentencia SQL que admita una tabla.
  - ✓ Por ejemplo, en la cláusula **FROM** de una sentencia **SELECT.**
- A la hora de crear una función en línea hay que tener en cuenta:
  - ✓ En la cláusula **RETURNS** debe especificarse el tipo **table.**
  - ✓ La cláusula **RETURN** debe contener una única sentencia **SELECT** entre paréntesis.
    - El conjunto de resultados de dicha **SELECT** constituye el valor-tabla que devuelve la función.
    - La sentencia **SELECT** que se utiliza en una **función en Línea** está sujeta a las mismas restricciones que las **SELECT** que se utilizan en una vista.

2.- TIPOS DE FUNCIONES: CON VALORES DE TABLA

- Una **función con Valores de Tabla** devuelve una tabla.
- Las **funciones con Valores de Tabla** son similares a los procedimientos.
  - ✓ La diferencia es que no tienen que devolver un número determinado de valores, sino que, al igual que las **funciones en Línea**, devuelven una tabla con un número indeterminado de filas.
- Al igual que ocurre con las **funciones en Línea**, puede hacerse referencia a una **función con Valores de Tabla** en cualquier lugar de una sentencia SQL que admita una tabla.
- A la hora de crear una **función con Valores de Tabla** hay que tener en cuenta:
  - ✓ La cláusula **RETURNS** define un nombre para la tabla y su formato.
    - El ámbito de dicho nombre es local a la función.
  - ✓ La cláusula **RETURN** debe especificarse sin ningún parámetro.

3.- DESENCADENADORES (TRIGGERS).

- Un desencadenador o **Trigger** es un tipo especial de procedimiento almacenado que se activa por sucesos en vez de por llamadas directas.

- Un **trigger** está asociado a una tabla y se ejecuta cuando una sentencia de actualización (**INSERT, UPDATE o DELETE**) modifica algún dato de dicha tabla.

- Los **triggers** y las sentencias que desencadenan su ejecución trabajan unidos como una única transacción, es decir, si la transacción no se ejecuta satisfactoriamente en toda su integridad, la BD vuelve a su estado inicial.

3.- DESENCADENADORES (TRIGGERS).

- **Los triggers se utilizan** para realizar, entre otras, las siguientes acciones:
  - ✓ Exigir integridad de datos más compleja que una restricción CHECK.
    - Esto es debido a que, a diferencia de la restricción **CHECK**, un **trigger**, aunque está asociado a una tabla, puede hacer referencia a columnas de otras tablas.
  - ✓ Cambios en cascada en varias tablas de una BD.
  - ✓ Definir mensajes de error personalizados.
    - Dado que las restricciones sólo pueden comunicar los errores a través de los mensajes estándar del sistema, si la aplicación requiere mensajes de error personalizados deben utilizarse **triggers**.

3.- DESENCADENADORES (TRIGGERS).

- **A la hora de utlizar triggers** hay que tener en cuenta lo siguiente:

  - ✓ En el momento de ejecutar una instrucción de actualización de datos en una tabla que lleve asociado un **trigger**, el sistema comprueba que se cumplen las restricciones definidas sobre la BD, en caso afirmativo, se ejecuta la instrucción de actualización y, posteriormente, el trigger.

  - ✓ Los triggers pueden definirse para una sola acción o para varias, asimismo, una tabla puede llevar asociados múltiples triggers.

  - ✓ Los triggers pueden hacer referencia a vistas y tablas temporales.

- **Cuando una instrucción de actualización de datos que invoca a un trigger afecta a varias filas**, el programador puede elegir entre:
  - ✓ Procesar todas las filas juntas, con lo que todas las filas afectadas deberán cumplir con los criterios especificados en el **trigger** para que se ejecute la acción.
  - ✓ Permitir acciones condicionales, de manera que la instrucción de actualización afecte únicamente a las filas que cumplan con los criterios especificados en el **trigger**.

3.1.- CREACIÓN DE TRIGGERS.

- **Sintaxis:**

```
CREATE TRIGGER NombreTrigger
        ON NombreTabla | NombreVista
        FOR  INSERT | UPDATE | DELETE
    AS
    <Sentencia-SQL> …
```

- **Nota**: Únicamente se puede especificar una vista cuando se utiliza la cláusula **INSTEAD OF**, que provoca que se ejecuten las acciones especificadas en el **trigger** en vez de la instrucción SQL desencadenadora. (Ver manual de referencia).

3.1.- CREACIÓN DE TRIGGERS: INSERT

- Opción **INSERT:**
  - ✓ Cuando se insertan una o más filas en una tabla, el sistema crea una tabla temporal, denominada **inserted**, con el mismo esquema que la tabla modificada, y almacena en ella una copia de la o las filas insertadas.
  - ✓ Si la tabla tiene un **trigger INSERT** asociado, éste puede examinar la tabla **inserted** para determinar qué acciones debe realizar y cómo ejecutarlas.

3.1.- CREACIÓN DE TRIGGERS: DELETE

- Opción **DELETE:**
  - ✓ Cuando se borran una ó más filas de una tabla, el sistema crea una tabla temporal, denominada **deleted**, con el mismo esquema que la tabla modificada y con una copia de la o las filas eliminadas.
  - ✓ Un trigger **DELETE** asociado a la tabla podrá examinar la tabla **deleted** para determinar qué acciones debe realizar y cómo ejecutarlas.

3.1.- CREACIÓN DE TRIGGERS: UPDATE

- Opción **UPDATE:**
  - ✓ Una sentencia **UPDATE** consta de dos acciones:
    - La acción **DELETE,** que borra de la tabla las filas a modificar.
    - La acción **INSERT,** que inserta en la tabla las filas modificadas.
  - ✓ Debido a lo anterior, cuando se ejecuta una instrucción **UPDATE** sobre una tabla se crean dos tablas temporales:
    - **deleted:** Almacena las filas originales.
    - **inserted:** Almacena las filas actualizadas.
  - ✓ Un **trigger UPDATE** asociado a dicha tabla podrá examinar ambas.

3.2.- MODIFICACIÓN Y BORRADO DE TRIGGERS.

- **Sintaxis de Modificación De Triggers:**

```
ALTER TRIGGER NombreTrigger
        ON NombreTabla | NombreVista
        FOR  INSERT | UPDATE | DELETE
    AS
    <Sentencia-SQL> …
```

- **Sintaxis de Eliminación De Triggers:**

```
DROP TRIGGER NombreTrigger
```

3.3.- HABILITACIÓN/DESHABILITACIÓN DE TRIGGERS

- **Sintaxis:**

ALTER TABLE NombreTabla | NombreVista
ENABLE | DISABLE TRIGGER
ALL | NombreTrigger1 [, …]

- ✓ Esta sentencia habilita|deshabilita uno o varios de los triggers asociados a una tabla o vista.
- ✓ Cuando se deshabilita un trigger, su definición se mantiene, pero la ejecución de una instrucción de actualización de datos no lo activa hasta que éste no se vuelva a habilitar.

3.4.- ANIDAMIENTO DE TRIGGERS

- Dado que en el cuerpo de un **trigger** puede incluirse una instrucción **INSERT**, **UPDATE** o **DELETE** que afecte a otra tabla, la ejecución de este trigger puede provocar la ejecución de otro trigger asociado a la tabla cuyo contenido va a ser modificado por el primero y así sucesivamente. Estaremos pues, ante **triggers anidados**.

3.4.- ANIDAMIENTO DE TRIGGERS

- A la hora de utilizar **desencadenadores anidados** hay que tener en cuenta que:
  - ✓ La opción de permitir la ejecución de triggers anidados está habilitada por defecto, pudiendo deshabilitarse con el procedimiento **sp_configure**
  - ✓ Un **trigger** no puede llamarse a sí mismo en respuesta a una segunda actualización de su tabla asociada, es decir, si un **trigger A** actualiza una tabla que tiene a su vez asociado un **trigger B** que provoca la actualización de la tabla a la que está asociado el **trigger A**, éste no se vuelve a activar.
    - Si queremos que un trigger pueda activarse a sí mismo de manera recursiva, debemos habilitar dicha opción mediante el procedimiento **sp_dboption.**
  - ✓ Dado que toda la cadena de **triggers** anidados forman una única transacción, si ésta no termina satisafactoriamente se deshacen todas las actualizaciones realizadas hasta ese momento.

3.5.- INSTRUCCIÓN IF UPDATE( )

- **Sintaxis:**

IF UPDATE (NombreColumna1) [{AND | OR UPDATE (NombreColumna2) } […]]

- ✓ Esta instrucción permite que un **trigger** se active únicamente cuando se actualicen datos de una ó más columnas específicas.
- ✓ Puede incluirse dentro del cuerpo de un **trigger INSERT** ó **UPDATE.**
- ✓ **UPDATE()** devolverá cierto en cualquier inserción de filas o modificación de datos que afecte a las columnas especificadas.