UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

LABORATORIO DE COMPUTACIÓN II

Unidad Temática N°3:
Objetos de Bases de Datos

Teórico
1° Año – 2° Cuatrimestre

Índice

Introducción a las Vistas

Una vista ofrece la posibilidad de almacenar una consulta predefinida como un objeto en una base de datos para usarse posteriormente. Las tablas consultadas en una vista se denominan tablas base. Algunos ejemplos habituales de vistas son los siguientes:

- Un subconjunto de las filas o columnas de una tabla base.
- Una unión de dos o más tablas base.
- Una combinación de dos o más tablas base.
- Un resumen estadístico de una tabla base.
- Un subconjunto de otra vista o alguna combinación de vistas y tablas base.

Ventajas de las vistas

Las vistas ofrecen diversas ventajas, como:

- Centrar el interés en los datos de los usuarios: Las vistas crean un entorno controlado que permite el acceso a datos específicos mientras se oculta el resto. Los usuarios pueden tratar la presentación de los datos en una vista de forma similar a como lo hacen en una tabla.
- Enmascarar la complejidad de la base de datos: Las vistas ocultan al usuario la complejidad del diseño de la base de datos. De este modo, los programadores pueden cambiar el diseño sin afectar a la interacción entre el usuario y la base de datos.
- Simplificar la administración de los permisos de usuario: En lugar de conceder a los usuarios permisos para consultar columnas específicas de las tablas base, los propietarios de las bases de datos pueden conceder permisos para que el usuario sólo pueda consultar los datos a través de vistas.
- Mejorar el rendimiento: Las vistas le permiten almacenar los resultados de consultas complejas. Otras consultas pueden utilizar estos resultados resumidos.
- Organizar los datos para exportarse a otras aplicaciones.

Creación de Vistas

Puede crear vistas con el Asistente para creación de vistas, con el Administrador corporativo de SQL Server o con Transact-SQL. Las vistas sólo se pueden crear en la base de datos actual. La sintaxis básica (parcial) para crear una vista es la siguiente:

```
create view NOMBREVISTA as
  SENTENCIASSELECT
   from TABLA;
```

En el siguiente ejemplo se crea la vista "vis_clientes", que es resultado de una combinación en la cual se muestran 4 campos:

```
create view vis_clientes
as
  select  (ape_cliente+' '+nom_cliente) as nombre, barrio,
          calle, altura
  from    clientes c
          join barrios b
          on c.cod_barrio=b.cod_barrio
```

La vista después de creada se trata como una tabla, es por ello que para ver el resultado de la misma se utiliza una sentencia select:

```
select * from vis_clientes;
```

Los campos y expresiones de la consulta que define una vista DEBEN tener un nombre. Se debe colocar nombre de campo cuando es un campo calculado o si hay 2 campos con el mismo nombre.

Los nombres de los campos y expresiones de la consulta que define una vista DEBEN ser únicos (no puede haber dos columnas en la vista con el mismo nombre o alias).

Se crea la vista "vis_clientes_activos" que consulta los clientes que registran compras en los últimos dos años (este año y el anterior):

```
create view vis_clientes_activos (nombre,barrio)
 as
   select  distinct ape_cliente+' '+nom_cliente, barrio
   from    clientes c
```

```
            join barrios b
            on c.cod_barrio=b.cod_barrio
            join facturas f
            on c.cod_cliente=f.cod_cliente
   Where    year(fecha)>=year(getdate())-1
```

La diferencia es que se colocan entre paréntesis los encabezados de las columnas que aparecerán en la vista. Los nombres que se colocan entre paréntesis deben ser tantos como los campos o expresiones que se definen en la vista.

Restricciones

Existen algunas restricciones para el uso de "create view", a saber:

- no se pueden crear vistas temporales ni crear vistas sobre tablas temporales.
- no se pueden asociar reglas ni valores por defecto a las vistas.
- no puede combinarse con otras instrucciones en un mismo lote.
- Las vistas no pueden hacer referencia a más de 1.024 columnas.

Vistas encriptadas

Se puede ver el texto que define una vista ejecutando el procedimiento almacenado del sistema "**sp_helptext**" seguido del nombre de la vista:

SP_HELPTEXT nombrevista;

Para ocultar el texto que define una vista se emplea "with encryption". Si se crea la vista anterior con su definición oculta no se podrá ver su texto al usar “sp_help”:

```
create view vis_clientes_activos (nombre,barrio)
  with encryption
 as
   select  distinct ape_cliente+' '+nom_cliente, barrio
   from    clientes c
           join barrios b
           on c.cod_barrio=b.cod_barrio
           join facturas f
           on c.cod_cliente=f.cod_cliente
```

```
Where   year(fecha)>=year(getdate())-1
```

Eliminar Vistas

Para quitar una vista se emplea:

DROP VIEW nombrevista

Si se elimina una tabla a la que hace referencia una vista, la vista no se elimina, hay que eliminarla explícitamente. Por ejemplo, para eliminar la vista denominada "vis_clientes":

```
drop view vis_clientes;
```

Modificar Vistas

Para modificar una vista se puede hacer con "alter view". En el ejemplo siguiente se altera vis_clientes_activos para agregar la calle y altura:

```
alter view vis_clientes_activos (nombre,barrio,calle,altura)
 as
   select  distinct ape_cliente+' '+nom_cliente, barrio,
           calle, altura
   from    clientes c
           join barrios b
           on c.cod_barrio=b.cod_barrio
           join facturas f
           on c.cod_cliente=f.cod_cliente
   Where   year(fecha)>=year(getdate())-1
```

Si crea una vista con "select *" y luego agrega campos a la estructura de las tablas involucradas, los nuevos campos no aparecerán en la vista; esto es porque los campos se seleccionan al ejecutar "create view"; debe alterar la vista.

Modificar datos de una tabla a través de vistas

Si se modifican los datos de una vista, se modifica la tabla base. Se puede insertar, actualizar o eliminar datos de una tabla a través de una vista, teniendo en cuenta lo siguiente, las modificaciones que se realizan a las vistas:

- no pueden afectar a más de una tabla consultada.
- no se pueden cambiar los campos resultado de un cálculo.
- pueden generar errores si afectan a campos a las que la vista no hace referencia.

Introducción a los Procedimientos Almacenados

Un procedimiento almacenado es un conjunto de instrucciones a las que se les da un nombre, que se almacena en el servidor. Permiten encapsular tareas repetitivas. SQL Server permite los siguientes tipos de procedimientos almacenados:

- **Del sistema**: están almacenados en la base de datos "master" y llevan el prefijo "sp_".
- **Locales**: los crea el usuario
- **Temporales**: pueden ser locales, cuyos nombres comienzan con un signo numeral (#), o globales, cuyos nombres comienzan con 2 signos numeral (##). Los procedimientos almacenados temporales locales están disponibles en la sesión de un solo usuario y se eliminan automáticamente al finalizar la sesión; los globales están disponibles en las sesiones de todos los usuarios.
- **Remotos** son una característica anterior de SQL Server.
- **Extendidos**: se implementan como bibliotecas de vínculos dinámicos (DLL, Dynamic-Link Libraries), se ejecutan fuera del entorno de SQL Server. Generalmente llevan el prefijo "xp_".

Un procedimiento almacenado puede hacer referencia a objetos que no existen al momento de crearlo. Los objetos deben existir cuando se ejecute el procedimiento almacenado.

Los procedimientos almacenados en SQL Server pueden:

- Contener instrucciones que realizan operaciones en la base de datos y llamar a otros procedimientos almacenados.
- Aceptar parámetros de entrada.
- Devolver un valor de estado a un procedimiento.

- Devolver varios parámetros de salida.

Ventajas:

- Comparten la lógica de la aplicación con las otras aplicaciones, con lo cual el acceso y las modificaciones de los datos se hacen en un solo sitio.
- Permiten realizar todas las operaciones que los usuarios necesitan evitando que tengan acceso directo a las tablas.
- Reducen el tráfico de red; en vez de enviar muchas instrucciones, los usuarios realizan operaciones enviando una única instrucción.
- Pueden encapsular la funcionalidad del negocio. Las reglas o directivas empresariales encapsuladas en los procedimientos almacenados se pueden cambiar en una sola ubicación.
- Apartar a los usuarios de la exposición de los detalles de las tablas de la base de datos.
- Proporcionar mecanismos de seguridad: los usuarios pueden obtener permiso para ejecutar un procedimiento almacenado incluso si no tienen permiso de acceso a las tablas o vistas a las que hace referencia.

Los procedimientos almacenados pueden hacer referencia a tablas, vistas, a funciones definidas por el usuario, a otros procedimientos almacenados y a tablas temporales.

Un procedimiento almacenado puede incluir cualquier cantidad y tipo de instrucciones, excepto: *create default, create procedure, create rule, create trigger y create view.*

Si un procedimiento almacenado crea una tabla temporal, dicha tabla sólo existe dentro del procedimiento y desaparece al finalizar el mismo. Lo mismo sucede con las variables.

Creación y ejecución de Procedimientos Almacenados

Para crear un procedimiento almacenado se emplea la instrucción "create procedure". La sintaxis básica parcial es:

CREATE PROCEDURE nombreprocedimiento
AS instrucciones;

Con las siguientes instrucciones se crea un procedimiento almacenado llamado "pa_articulos_precios_mas100" que muestra todos los artículos de la librería con precios mayores a 100:

```
CREATE PROC pa_ articulos_precios_mas100
 AS
  SELECT *
  FROM   articulos
  WHERE  pre_unitario >100;
```

El procedimiento se ejecuta colocando "execute" (o "exec") seguido del nombre del procedimiento almacenado creado anteriormente como se muestra en la siguiente sentencia:

```
EXEC pa_articulos_precios_mas100;
```

Modificar procedimientos almacenados

Los procedimientos almacenados pueden modificarse, por necesidad de los usuarios o por cambios en la estructura de las tablas que referencia. Sintaxis:

ALTER PROCEDURE nombreprocedimiento
@parametro TIPO = valorpredeterminado
AS sentencias;

Modificar el procedimiento almacenado anterior para que muestre determinadas columnas:

```
ALTER PROC pa_articulos_precios_mas100
 AS
  SELECT descripcion, pre_unitario,observaciones,stock
  FROM   articulos
  WHERE  pre_unitario >100;
```

Eliminar procedimientos almacenados

Los procedimientos almacenados se eliminan con "DROP PROCEDURE". Sintaxis:

DROP PROCEDURE nombreprocedimiento;

Eliminar el procedimiento almacenado llamado "pa_articulosA":

```
DROP PROCEDURE pa_articulosA;
```

Procedimientos almacenados con parámetros de entrada

Los parámetros de entrada posibilitan pasar información a un procedimiento. Para que un procedimiento almacenado admita parámetros de entrada se deben declarar variables como parámetros al crearlo. La sintaxis es:

CREATE PROC nombreprocedimiento
@nombreparametro tipo =valorpordefecto
AS sentencias;

Los parámetros se definen luego del nombre del procedimiento, comenzando con un signo arroba (@). Los parámetros son locales al procedimiento, es decir, existen solamente dentro del mismo. Pueden declararse varios parámetros por procedimiento, se separan por comas.

Cuando el procedimiento es ejecutado, deben explicitarse valores para cada uno de los parámetros (en el orden que fueron definidos), a menos que se haya definido un valor por defecto, en tal caso, pueden omitirse.

Luego de definir un parámetro y su tipo, se puede especificar un valor por defecto; tal valor es el que asume el procedimiento al ser ejecutado si no recibe parámetros. El valor por defecto puede ser "null" o una constante, también puede incluir comodines si el procedimiento emplea "like".

Como ejemplo se crea un procedimiento que recibe dos precios unitarios para mostrar todos aquellos que se encuentren entre dichos valores:

```
create procedure pa_articulos_precios
 @precio1 money,
 @precio2 money,
AS
 select  descripcion,pre_unitario,observaciones
  from   articulos
  where  pre_unitario between @precio1 and @precio2;
```

El procedimiento se ejecuta colocando "exec" seguido del nombre del procedimiento y los valores para los parámetros separados por comas:

```
exec pa_articulos_precios 100,200;
```

El ejemplo anterior ejecuta el procedimiento pasando valores a los parámetros por posición. También podemos emplear la otra sintaxis en la cual pasamos valores a los parámetros por su nombre:

```
exec pa_articulos_precios @precio1=100, @precio2=200;
```

En este caso, cuando los valores son pasados con el nombre del parámetro, el orden en que se colocan puede alterarse.

```
exec pa_articulos_precios @precio2=200, @precio1=100;
```

Si se quiere ejecutar un procedimiento que permita omitir los valores para los parámetros se debe, al crear el procedimiento, definir valores por defecto para cada parámetro:

```
create procedure pa_articulos_precios
  @precio1 money=100,
  @precio2 money=200,
 AS
  select descripcion,pre_unitario,observaciones
   from articulos
   where pre_unitario between @precio1 and @precio2;


exec pa_articulos_precios;
```

Si se envía un solo parámetro a un procedimiento que tiene definido más de un parámetro sin especificar a qué parámetro corresponde (valor por posición), asume que es el primero.

Se puede emplear patrones de búsqueda en la consulta que define el procedimiento almacenado y utilizar comodines como valores por defecto:

```
 create procedure pa_articulos_descripcion
  @descripcion varchar(100)= '%'
AS
  select descripcion,pre_unitario,observaciones
   from articulos
   where descripción like @descripcion;
```

```
exec pa_articulos_descripcion;
```

Procedimientos almacenados con parámetros de salida

Dijimos que los procedimientos almacenados pueden devolver información; para ello se emplean parámetros de salida. El valor se retorna a quien realizó la llamada con parámetros de salida.

Para que un procedimiento almacenado devuelva un valor se debe declarar una variable con la palabra clave "output" al crear el procedimiento:

```
CREATE PROCEDURE nombreprocedimiento
  @parametroentrada tipo =valorpordefecto,
  @parametrosalida tipo=valorpordefecto output
 AS
   sentencias
   select @parametrosalida=sentencias;
```

Los parámetros de salida pueden ser de cualquier tipo de datos, excepto text, ntext e image. Se crea un procedimiento almacenado que muestre los descripción, precio y observaciones de los articulos que comiencen con una (o más) letra/s determinada/s (enviada como parámetro de entrada) y retorne la suma y el promedio de los precios de todos los artículos que cumplan con la condición de búsqueda:

```
create procedure pa_articulos_sumaypromedio
 @descripcion varchar(100)='%',
 @suma decimal(10,2) output,
 @promedio decimal(8,2) output
 as
  select descripcion,pre_unitario,observaciones
   from  articulos
   where descripción like @descripcion
 select @suma=sum(precio)
  from  articulos
  where descripción like @descripcion
 select @promedio=avg(precio)
  from  articulos
  where descripción like @descripcion;
```

Al ejecutar el procedimiento se puede ver el contenido de las variables en las que se almacenan los parámetros de salida del procedimiento (Al ejecutarlo también debe emplearse "output"):

```
declare @s decimal(10,2), @p decimal(8,2)
execute pa_autor_sumaypromedio 'lápiz%', @s output, @p output
select @s as total, @p as promedio;
```

Procedimientos almacenados: return

La instrucción "return" sale de una consulta o procedimiento y todas las instrucciones posteriores no son ejecutadas.

Como ejemplo se crea un procedimiento que muestre todos los articulos de una descripción determinada que se ingresa como parámetro:

```
create procedure pa_articulos_descrip
 @descripcion varchar(100)= null
as
if @descripcion is null
begin
 select 'Debe indicar una descripción'
 return
end;
  select descripcion, pre_unitario, observaciones
  from articulos
  where descripción like @descripcion
```

Si al ejecutar el procedimiento enviamos el valor "null" o no pasamos valor, con lo cual toma el valor por defecto "null", se muestra un mensaje y sale; en caso contrario, ejecuta la consulta luego del "else"; "return" puede retornar un valor entero.

Se crea un procedimiento almacenado que ingresa registros en la tabla "articulos". Los parámetros correspondientes a la descripción y precio DEBEN ingresarse con un valor distinto de "null", los demás son opcionales. El procedimiento retorna "1" si la inserción se realiza, es decir, si se ingresan valores para descripción y precio y "0", en caso que descripción y precio sean nulos:

```
create procedure pa_articulos_alta
 @descrip varchar(100)=null,
 @precio money=null,
 @observ varchar(100)=null
as
 if (@descrip is null) or (@precio is null)
  return 0
 else
 begin
 insert into articulos(descripcion,pre_unitario,observaciones)
```

```
 values (@descrip,@precio,@observ)
return 1
end;
```

Para ver el resultado, debemos declarar una variable en la cual se almacene el valor devuelto por el procedimiento; luego, ejecutar el procedimiento asignándole el valor devuelto a la variable, finalmente mostramos el contenido de la variable:

```
declare @retorno int
exec @retorno=pa_articulos_alta 'Lapicera Big azul',45,'trazo fino'
select 'Ingreso realizado=1' = @retorno
exec @retorno=pa_articulos_alta
select 'Ingreso realizado=1' = @retorno;
```

También podríamos emplear un "if" para controlar el valor de la variable de retorno:

```
declare @retorno int;
exec @retorno= pa_articulos_alta 'Cuaderno 24 hojas',60
if @retorno=1 print 'Registro ingresado'
else select 'Registro no ingresado porque faltan datos';
```

Procedimientos almacenados encriptados

Si no quiere que los usuarios puedan leer el contenido del procedimiento podemos indicarle a SQL Server que codifique la entrada a la tabla "syscomments" que contiene el texto. Para ello, debemos colocar la opción "with encryption" al crear el procedimiento:

```
CREATE PROCEDURE nombreprocedimiento
 parámetros
 WITH ENCRYPTION
 AS instrucciones;
```

Si se ejecuta el procedimiento almacenado del sistema "sp_helptext" para ver su contenido, no aparece.

Procedimientos almacenados: Insert

Podemos ingresar datos en una tabla con el resultado devuelto por un procedimiento almacenado. La instrucción siguiente crea el procedimiento "pa_ofertas", que ingresa libros en la tabla "ofertas":

```
 create proc pa_articulos_ofertas
 as
select descripcion,pre_unitario,observaciones
   from articulos
   where pre_unitario<100;
```

La siguiente instrucción ingresa en la tabla “ofertas” el resultado del procedimiento “pa_ofertas”:

```
 insert into ofertas exec pa_articulos_ofertas;
```

Las tablas deben existir y los tipos de datos deben coincidir.

Anidamiento de procedimientos almacenados

Los procedimientos almacenados pueden anidarse, es decir, un procedimiento almacenado puede llamar a otro. Las características del anidamiento de procedimientos almacenados son las siguientes:

- Los procedimientos almacenados se pueden anidar hasta 32 niveles.
- El nivel actual de anidamiento se almacena en la función del sistema @@nestlevel.
- Si un procedimiento almacenado llama a otro, éste puede obtener acceso a todos los objetos que cree el primero, incluidas las tablas temporales.
- Los procedimientos almacenados anidados pueden ser recursivos. Por ejemplo, el procedimiento almacenado A puede llamar al procedimiento almacenado B y éste puede llamar al procedimiento almacenado A.

Se creamos un procedimiento que retorne el factorial de un número:

```
create procedure pa_factorial
 @numero int
as
 declare @resultado int
 declare @num int
 set @resultado=1
 set @num=@numero
 while (@num>1)
 begin
```

```
    exec pa_multiplicar @resultado,@num, @resultado output
    set @num=@num-1
  end
  select rtrim(convert(char,@numero))+'!='+convert(char,@resultado);
```

Cuando un procedimiento (A) llama a otro (B), el segundo (B) tiene acceso a todos los objetos que cree el primero (A).

Funciones definidas por el usuario

Con Microsoft SQL Server, puede diseñar sus propias funciones para complementar y ampliar las funciones (integradas) suministradas por el sistema. SQL Server admite tres tipos de funciones definidas por el usuario:

- **Funciones escalares**: es similar a una función integrada.
- **Funciones con valores de tabla de varias instrucciones**: devuelve una tabla creada por una o varias instrucciones Transact-SQL y es similar a un procedimiento almacenado.
- **Funciones con valores de tabla en línea**: devuelve una tabla que es el resultado de una sola instrucción SELECT.

Funciones Escalares

Una función escalar retorna un único valor. Como todas las funciones, se crean con la instrucción "create function". La sintaxis básica es:

```
CREATE FUNCTION nombre
(@parametro tipo=valorpordefecto)
 RETURNS tipo
 BEGIN
  instrucciones
  RETURN valor
 END;
```

Luego del nombre se colocan (opcionalmente) los parámetros de entrada con su tipo. La cláusula “returns” indica el tipo de dato retornado. El cuerpo de la función, se define en un bloque “begin…end” que contiene las instrucciones que retornan el valor. Creamos una función denominada “f_promedio” que recibe 2 valores y retorna el promedio:

```
create function f_promedio
(@valor1 decimal(4,2),
 @valor2 decimal(4,2)
)
```

```
returns decimal (6,2)
as
begin
  declare @resultado decimal(6,2)
  set @resultado=(@valor1+@valor2)/2
  return @resultado
end;
```

En el ejemplo anterior se declara una variable local a la función (desaparece al salir de la función) que calcula el resultado que se retornará. Al hacer referencia a una función escalar, se debe especificar el propietario y el nombre de la función:

```
select dbo.f_promedio(5.5,8.5);
```

Cuando se llama a funciones que tienen definidos parámetros de entrada SE DEBEN suministrar SIEMPRE un valor para él.

Crear una función a la cual se le envía una fecha y retorna el nombre del mes en español:

```
create function f_nombreMes
 (@fecha datetime='2007/01/01')
  returns varchar(10)
  as
  begin
    declare @nombre varchar(10)
    set @nombre=
     case datename(month,@fecha)
       when 'January' then 'Enero'
       when 'February' then 'Febrero'
       when 'March' then 'Marzo'
       when 'April' then 'Abril'
       when 'May' then 'Mayo'
       when 'June' then 'Junio'
       when 'July' then 'Julio'
       when 'August' then 'Agosto'
       when 'September' then 'Setiembre'
       when 'October' then 'Octubre'
       when 'November' then 'Noviembre'
       when 'December' then 'Diciembre'
     end--case
    return @nombre
 end;
```

Las funciones que retornan un valor escalar pueden emplearse en cualquier consulta donde se coloca un campo.

```
select nombre,
  dbo.f_nombreMes(fechaingreso) as 'mes de ingreso'
 from empleados;
```

Se puede colocar un valor por defecto al parámetro, pero al invocar la función, para que tome el valor por defecto SE DEBE especificar "default".

```
select dbo.f_nombreMes(default);
```

La instrucción "create function" debe ser la primera sentencia de un lote.

Funciones de tabla de varias instrucciones

Las funciones que retornan una tabla pueden emplearse en lugar de un "from" de una consulta. Sintaxis:

```
CREATE FUNCTION nombrefuncion
(@parametro tipo)
RETURNS @nombretablaretorno TABLE-- nombre de la tabla
--formato de la tabla
(campo1 tipo,
 campo2 tipo,
 campo3 tipo
)
AS
BEGIN
  INSERT @nombretablaretorno
   SELECT campos
    FROM tabla
    WHERE campo operador @parametro
  RETURN
END
```

La cláusula "returns" define un nombre de variable local para la tabla que retornará, el tipo de datos a retornar (que es "table") y el formato de la misma (campos y tipos).

El siguiente ejemplo crea una función denominada "f_ofertas" que recibe un parámetro. La función retorna una tabla con el codigo, descripcion pre_unitario de todos los articulos cuyo precio sea inferior al parámetro:

```
create function f_ofertas
(@minimo decimal(8,2))
returns @ofertas table-- nombre de la tabla
--formato de la tabla
```

```
(cod_articulo int,
 descripcion varchar(100),
 pre_unitario money,
 observaciones varchar(100)
)
as
begin
  insert @ofertas
   select cod_articulo,descripcion,pre_unitario,observaciones
   from articulos
   where pre_unitario < @minimo
  return
end;
```

La siguiente consulta realiza un join entre la tabla "articulos" y la tabla retornada por la función "f_ofertas":

```
select *
from  articulos as a
 join dbo.f_ofertas(25) as o
 on a.cod_articulo=o.articulo;
```

Se puede llamar a la función como si fuese una tabla o vista listando algunos campos:

```
select descripcion,pre_unitario from dbo.f_ofertas(40);
```

Funciones con valores de tabla en línea

Una función con valores de tabla en línea retorna una tabla que es el resultado de una única instrucción "select". Es similar a una vista, pero con la posibilidad de utilizar parámetros. Sintaxis:

```
CREATE FUNCTION nombrefuncion
(@parametro tipo=valorpordefecto)
RETURNS TABLE
AS
RETURN (
 SELECT campos
 FROM tabla
 WHERE condición
);
```

El valor por defecto es opcional.

"returns" especifica "table" como el tipo de datos a retornar. No se define el formato de la tabla a retornar porque queda establecido en el "select".

El cuerpo de la función no contiene un bloque "begin...end" como las otras funciones.

La cláusula "return" contiene una sola instrucción "select" entre paréntesis. El resultado del "select" es la tabla que se retorna. El "select" está sujeto a las mismas reglas que los "select" de las vistas.

Se crea una función con valores de tabla en línea que recibe un valor de autor como parámetro:

```
create function f_articulos
(@descrip varchar(100)='Lápiz')
returns table
as
return (
 select descripcion,pre_unitario
 from articulos
 where descripcion like '%'+@descrip+'%'
);
```

Estas funciones retornan una tabla y se hace referencia a ellas en la cláusula "from", como una vista:

```
select * from f_articulos('Lápiz');
```

Funciones encriptadas

Las funciones definidas por el usuario pueden encriptarse, para evitar que sean leídas con "sp_helptext". Para ello debemos agregar al crearlas la opción "with encryption" antes de "as". En funciones escalares. En funciones de tabla de varias sentencias se coloca luego del formato de la tabla a retornar.

Modificar Funciones

Las funciones definidas por el usuario pueden modificarse con la instrucción "alter function".

Sintaxis para modificar funciones escalares:

ALTER FUNTION propietario.nombrefuncion

```
(@parametro tipo=valorpordefecto)
 RETURNS tipo
 AS
 BEGIN
  cuerpo
  RETURN expresionescalar
 END
```

Sintaxis para modificar una función de varias instrucciones que retorna una tabla:

```
ALTER FUNCTION NOMBREFUNCION
 (@parametro tipo=valorpordefecto)
 RETURNS @variable TABLE
 (definicion de la tabla a retornar)
 AS
 BEGIN
   cuerpo de la funcion
   RETURN
 END
```

Sintaxis para modificar una función con valores de tabla en línea

```
ALTER FUNCTION nombrefuncion
(@parametro tipo)
RETURNS TABLE
AS
 RETURN (sentencias select)
```

En un ejemplo: se crea una función que retorna una tabla en línea:

```
create function f_articulos
(@descrip varchar(100)='Lápiz')
returns table
as
return (
 select descripcion,pre_unitario
 from articulos
 where descripcion like '%'+@descrip+'%'
);
```

Para modificar la función agregando otro campo en el "select":

```
alter function f_articulos
(@descrip varchar(100)='Lápiz')
returns table
as
return (
 select descripcion,pre_unitario,observaciones
 from articulos
 where descripcion like '%'+@descrip+'%'
);
```

Eliminar Funciones

Las funciones definidas por el usuario se eliminan con la instrucción “drop function”: Sintaxis:

DROP FUNCTION nombreppropietario.nombrefuncion;

Se coloca el nombre del propietario seguido del nombre de la función.

Si la función que se intenta eliminar no existe, aparece un mensaje indicándolo, para evitarlo, podemos verificar su existencia antes de solicitar su eliminación

Se elimina, si existe, la función denominada "f_fechacadena":

```
if object_id('dbo.f_fechacadena') is not null
 drop function dbo.f_fechacadena;
```

BIBLIOGRAFÍA

Gorman K., Hirt A., Noderer D., Rowland-Jones J., Sirpal A., Ryan D. & Woody B (2019) Introducing Microsoft SQL Server 2019. Reliability, scalability, and security both on premises and in the cloud. Packt Publishing Ltd. Birmingham UK

Microsoft (2021) SQL Server technical documentation. Disponible en: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15

Opel, A. & Sheldon, R. (2010). Fundamentos de SQL. Madrid. Editorial Mc Graw Hill

Varga S., Cherry D., D'Antoni J. (2016). Introducing Microsoft SQL Server 2016 Mission-Critical Applications, Deeper Insights, Hyperscale Cloud. Washington. Microsoft Press