UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

BASES DE DATOS I

Unidad Temática N°3:
Programación en SQL Server

Guía de estudio
1° Año – 2° Cuatrimestre

Índice

Problema 3.1: Introducción a la Programación en SQL Server

Utilizando la base de datos Librería realizar las consignas detalladas a continuación.

1. **Declarar 3 variables que se llamen codigo, stock y stockMinimo respectivamente. A la variable codigo setearle un valor. Las variables stock y stockMinimo almacenarán el resultado de las columnas de la tabla artículos stock y stockMinimo respectivamente filtradas por el código que se corresponda con la variable codigo.**

2. **Utilizando el punto anterior, verificar si la variable stock o stockMinimo tienen algún valor. Mostrar un mensaje indicando si es necesario realizar reposición de artículos o no.**

3. **Modificar el ejercicio 1 agregando una variable más donde se almacene el precio del artículo. En caso que el precio sea menor a $500, aplicarle un incremento del 10%. En caso de que el precio sea mayor a $500 notificar dicha situación y mostrar el precio del artículo.**

4. **Declarar dos variables enteras, y mostrar la suma de todos los números comprendidos entre ellos. En caso de ser ambos números iguales mostrar un mensaje informando dicha situación**

5. **Mostrar nombre y precio de todos los artículos. Mostrar en una tercer columna la leyenda 'Muy caro' para precios mayores a $500, 'Accesible' para precios entre $300 y $500, 'Barato' para precios entre $100 y $300 y 'Regalado' para precios menores a $100.**

6. **Modificar el punto 2 reemplazando el mensaje de que es necesario reponer artículos por una excepción.**

Problema 3.2: Procedimientos Almacenados

Desde el área de programación del sistema de información de la librería nos solicitan la creación de objetos en la base de datos que permitan ingresar parámetros en el momento de la ejecución de los mismos, dichos objetos en SQL Server son denominados Procedimientos Almacenados:

Para crear un procedimiento almacenado empleamos:

```
CREATE PROCEDURE nombreprocedimiento
 AS instrucciones;
```

Con las siguientes instrucciones se crea un procedimiento almacenado llamado "pa_aarticulos_precios" que muestra todos los articulos cuyo precio unitario es menor a 100:

```
CREATE PROC pa_articulos_precios
 AS
  SELECT * FROM articulos
  WHERE pre_unitario <100;
```

Para ejecutar el procedimiento almacenado creado anteriormente tipeamos:

```
EXEC pa_articulos_precios;
```

Los procedimientos almacenados se eliminan con "DROP PROCEDURE".

Para eliminar el procedimiento almacenado llamado " pa_articulos_precios ":

```
DROP PROCEDURE pa_articulos_precios;
```

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. 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 luego de definir un parámetro y su tipo; 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".

Se crea, en el ejemplo un procedimiento que muestre los artículos cuyo precio unitario sea menor a un precio que se ingresará en el momento en que se ejecute el mismo:

```
CREATE PROC pa_articulos_precios2
@precio money
  AS
   SELECT * FROM articulos
```

```
   WHERE pre_unitario <@precio
```

Luego se ejecuta con la siguiente sentencia:

```
exec pa_articulos_precios2 100';
```

Si se quiere listar los artículos cuyo precio esté entre dos valores.

```
CREATE PROC pa_articulos_precios3
@precio1 money,
@precio2 money
  AS
   SELECT * FROM articulos
   WHERE pre_unitario between @precio1 and @precio2

exec pa_ articulos_precios3 @precio1=50, @precio2=100;
```

Cuando se pasa valores con el nombre del parámetro, el orden en que se colocan puede alterarse.

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;
```

Crear un procedimiento almacenado que muestre la descripción de un artículo de código determinado (enviado como parámetro de entrada) y nos retorne el total facturado para ese artículo y el promedio ponderado de los precios de venta de ese artículo

```
 Create proc pa_ventas_articulo
@codigo int,
@total decimal(10,2) output,
@precioprom decimal(6,2) output
as
 select descripcion from articulos
 where cod_articulo=@codigo
 select @total=sum(cantidad*pre_unitario)
 from detalle_facturas
 where cod_articulo=@codigo
 select @precioprom=sum(pre_unitario)/sum(cantidad)
 from detalle_facturas
 where cod_articulo=@codigo
```

Ejecutar el procedimiento:

```
 declare @s decimal(12,2), @p decimal(10,2)
execute pa_ventas_articulo 5, @s output, @p output
select @s total, @p  'Precio promedio'
```

Crear un procedimiento que muestre todos los libros de un autor determinado que se ingresa como parámetro:

```
create procedure pa_articulos_precios
 @precio decimal(12,2)=null
as
if @precio is null
begin
 select 'Debe indicar un precio'
 return
end;

select descripcion from  articulos where precio< @precio
```

Crear un procedimiento almacenado que ingresa registros en la tabla "articulos". Los parámetros correspondientes al pre_unitario 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 pre_unitario y "0", en caso que pre_unitario sea nulo:

```
create procedure pa_articulos_ingreso
@descripcion nvarchar (50) NULL ,
@stock_minimo smallint NULL ,
@pre_unitario decimal(10, 2) NOT NULL ,
@observaciones nvarchar (50)=null,
as
 if (@pre_unitario is null)
  return 0
 else
 begin
  insert into articulos
values(@descripcion,@stock_minimo,@pre_unitario,@observaciones)
  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_ingreso     @descripcion     ='Regla
50cm',@pre_unitario=75
 select 'Ingreso realizado=1' = @retorno
```

```
exec @retorno=pa_articulos_ingreso
select 'Ingreso realizado=1' = @retorno;
```

1. **Cree los siguientes SP:**

    a. **Detalle_Ventas: liste la fecha, la factura, el vendedor, el cliente, el artículo, cantidad e importe. Este SP recibirá como parámetros de E un rango de fechas.**

    b. **CantidadArt_Cli : este SP me debe devolver la cantidad de artículos o clientes (según se pida) que existen en la empresa.**

    c. **INS_Vendedor: Cree un SP que le permita insertar registros en la tabla vendedores.**

    d. **UPD_Vendedor: cree un SP que le permita modificar un vendedor cargado.**

    e. **DEL_Vendedor: cree un SP que le permita eliminar un vendedor ingresado.**

2. **Modifique el SP 1-a, permitiendo que los resultados del SP puedan filtrarse por una fecha determinada, por un rango de fechas y por un rango de vendedores; según se pida.**

3. **Ejecute los SP creados en el punto 1 (todos).**

4. **Elimine los SP creados en el punto 1.**

5. **Programar procedimientos almacenados que permitan realizar las siguientes tareas:**
    a. **Mostrar los artículos cuyo precio sea mayor o igual que un valor que se envía por parámetro.**
    b. **Ingresar un artículo nuevo, verificando que la cantidad de stock que se pasa por parámetro sea un valor mayor a 30 unidades y menor que 100. Informar un error caso contrario.**
    c. **Mostrar un mensaje informativo acerca de si hay que reponer o no stock de un artículo cuyo código sea enviado por parámetro**
    d. **Actualizar el precio de los productos que tengan un precio menor a uno ingresado por parámetro en un porcentaje que también se envíe por parámetro. Si no se modifica ningún elemento informar dicha situación**
    e. **Mostrar el nombre del cliente al que se le realizó la primer venta en un parámetro de salida.**
    f. **Realizar un select que busque el artículo cuyo nombre empiece con un valor enviado por parámetro y almacenar su nombre en un parámetro de**

salida. En caso que haya varios artículos ocurrirá una excepción que deberá ser manejada con try catch.

Problema 3.3: Funciones definidas por el usuario

6. **Cree las siguientes funciones:**
    a. **Hora: una función que les devuelva la hora del sistema en el formato HH:MM:SS (tipo carácter de 8).**
    b. **Fecha: una función que devuelva la fecha en el formato AAAMMDD (en carácter de 8), a partir de una fecha que le ingresa como parámetro (ingresa como tipo fecha).**
    c. **Dia_Habil: función que devuelve si un día es o no hábil (considere como días no hábiles los sábados y domingos). Debe devolver 1 (hábil), 0 (no hábil)**
7. **Modifique la f(x) 1.c, considerando solo como día no hábil el domingo.**
8. **Ejecute las funciones creadas en el punto 1 (todas).**
9. **Elimine las funciones creadas en el punto 1.**
10. **Programar funciones que permitan realizar las siguientes tareas:**
    a. **Devolver una cadena de caracteres compuesto por los siguientes datos: Apellido, Nombre, Telefono, Calle, Altura y Nombre del Barrio, de un determinado cliente, que se puede informar por codigo de cliente o email.**
    b. **Devolver todos los artículos, se envía un parámetro que permite ordenar el resultado por el campo precio de manera ascendente (‘A’), o descendente (‘D’).**
    c. **Crear una función que devuelva el precio al que quedaría un artículo en caso de aplicar un porcentaje de aumento pasado por parámetro.**

Problema 3.4: Triggers

1. **Crear un desencadenador para las siguientes acciones:**
    a. **Restar stock DESPUES de INSERTAR una VENTA**
    b. **Para no poder modificar el nombre de algún artículo**
    c. **Insertar en la tabla HistorialPrecio el precio anterior de un artículo si el mismo ha cambiado**
    d. **Bloquear al vendedor con código 4 para que no pueda registrar ventas en el sistema.**

Problema 3.5: Manejo de errores

1. **Modificar el ejercicio 2 del problema 3.1 reemplazando los mensajes mostrados en consola con print, por excepciones. Verificar el comportamiento en el SQL Server Management.**

2. **Modificar el ejercicio anterior agregando las cláusulas de try catch para manejo de errores, y mostrar el mensaje capturado en la excepción con print.**

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