Resultados Unidad 4

Guía 4 Página 2 Ejercicio 4.1.1

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.

```
DECLARE @codigo INT
DECLARE @stock INT
DECLARE @stockMinimo INT

SET @codigo = 2;

SET @stock = (SELECT a.stock FROM articulos a where a.cod_articulo=
@codigo)
SET @stockMinimo = (select a.stock_minimo from articulos a where
a.cod_articulo= @codigo)
select @stock as 'stock'
select @stockMinimo as 'Stock minimo'
select * from articulos
```

Guía 4 Página 2 Ejercicio 4.1.2

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.

```
DECLARE @codigo INT
DECLARE @stock INT
DECLARE @stockMinimo INT

SET @codigo = 2;

SET @stock = (SELECT a.stock FROM articulos a where a.cod_articulo=
@codigo)
SET @stockMinimo = (select a.stock_minimo from articulos a where
a.cod_articulo= @codigo)
if @stock < @stockMinimo
begin
select 'Hay que reponer el stock'
end
else
select 'todo en orden'
```

Guía 4 Página 2 Ejercicio 4.1.3

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.

```
DECLARE @codigo INT
DECLARE @stock INT
DECLARE @stockMinimo INT
DECLARE @precio numeric(10,2)

--igualar a 1 para <500
--igualar a 2 para =500
--igualar a 3 para >500
SET @codigo = 3;

SET @stock = (select stock from articulos where cod_articulo=@codigo)
SET @stockMinimo = (select stock_minimo from articulos where cod_articulo=@codigo)
SET @precio = (select pre_unitario from articulos where cod_articulo=@codigo)


if(@precio <= 500)
begin
declare @precio_incrementado numeric(10,2)
set @precio_incrementado = @precio * 1.1
select @codigo'codigo', @stock'stock',@stockMinimo'stock minimo',@precio_incrementado'precio
actualizado'
end
else
begin
select @precio'precio articulo', 'El precio del articulo es mayor a $500' 'notificacion'
end
```

Guía 4 Página 2 Ejercicio 4.1.4

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

```
create FUNCTION F_suma (@numero1 INT, @numero2 INT)
     RETURNS VARCHAR(50)
     AS
     BEGIN
         DECLARE @suma INT
         SET @SUMA = 0
         DECLARE @numero INT

         IF(@numero1 = @numero2)
             RETURN 'Los numeros son iguales'

         SET @numero = @numero1
         WHILE (@numero <= @numero2)
         BEGIN
             SET @suma += @numero
             SET @numero += 1
         END
         RETURN CAST(@suma AS VARCHAR)
     END

SELECT dbo.F_suma(5, 5)
```

Guía 4 Página 2 Ejercicio 4.1.5

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.

```
SELECT descripcion,
        pre_unitario,
           tipoprecio=
CASE
WHEN pre_unitario>500 THEN 'Muy caro'
WHEN pre_unitario >= 300 AND pre_unitario <=500 THEN 'Accesible'
WHEN pre_unitario >= 100 AND pre_unitario <300 THEN 'Barato'
WHEN pre_unitario <100 THEN 'regalado'
END
FROM articulos
```

Guía 4 Página 2 Ejercicio 4.1.6

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

```
create proc f_codigoYstock2
@codigo int,
@stock int null output,
@stockMinimo int null output
as
begin
set @stock = (select stock from articulos where cod_articulo = @codigo)
set @stockMinimo = (select stock_minimo from articulos where
cod_articulo = @codigo)

begin try
if (@stock<@stockMinimo)
print 'debe reponer el articulo'
end try
begin catch
select 'se produjo el siguiente error',ERROR_NUMBER() numero,
ERROR_MESSAGE() mensaje,ERROR_STATE() estado,'habla con el administrador'
responsable
end catch
end

declare @sto int, @stock_mini int
execute f_codigoYstock2 4,@sto output,@stock_mini output
select @sto 'stock',@stock_mini 'stock minimo'
```

Guía 4 Página 2 Ejercicio 4.2.1

Modificar el ejercicio 2 de la sección 1.1 reemplazando los mensajes mostrados en consola con print, por excepciones. Verificar el comportamiento en el SQL Server Management.

```
DECLARE @codigo INT
DECLARE @stock INT
DECLARE @stockMinimo INT

SET @codigo = 2;

SET @stock = (SELECT a.stock FROM articulos a where a.cod_articulo=
@codigo)
SET @stockMinimo = (select a.stock_minimo from articulos a where
a.cod_articulo= @codigo)
if @stock < @stockMinimo
begin
select 'Hay que reponer el stock'
end
else
select 'todo en orden'


begin try
        if @stock is not null and @stockMinimo is not null
        begin
                if @stock < @stockMinimo
                begin
                        print '-- Reponer Stock --'
                end
                else
                        print '-- Stock aceptable --'
        end
        else
                print '-- Variable/s vacias --'
end try
begin catch
        select ERROR_MESSAGE() as Error
end catch
```

Guía 4 Página 2 Ejercicio 4.2.2

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

```
DECLARE @codigo INT
DECLARE @stock INT
DECLARE @stockMinimo INT

SET @codigo = 2;

SET @stock = (SELECT a.stock FROM articulos a where a.cod_articulo=
@codigo)
SET @stockMinimo = (select a.stock_minimo from articulos a where
a.cod_articulo= @codigo)
if @stock < @stockMinimo
begin
select 'Hay que reponer el stock'
end
else
select 'todo en orden'


begin try
        if @stock is not null and @stockMinimo is not null
        begin
                if @stock < @stockMinimo
                begin
                        print '-- Reponer Stock --'
                end
                else
                        print '-- Stock aceptable --'
        end
        else
                print '-- Variable/s vacias --'
end try
begin catch
        select ERROR_MESSAGE() as Error
end catch
```

Guía 4 Página 3 Ejercicio 4.3.1.A

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.

```
CREATE PROCEDURE SP_ARTICULOS_POR_RANGO
        @precio decimal(12,2) = null
AS
BEGIN
        if(@precio is null) print 'Debe ingresar un precio'
        else select descripcion 'Articulo',pre_unitario 'Precio' from articulos
where pre_unitario >= @precio
END

exec  SP_ARTICULOS_POR_RANGO 200
```

Guía 4 Página 3 Ejercicio 4.3.1.B

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.

```
CREATE PROCEDURE SP_431B

@STOCK int

AS

IF (@STOCK between 30 and 100)

        INSERT INTO articulos(descripcion, stock_minimo, stock, pre_unitario)
values('Goma', 20, @STOCK, 15.0)

ELSE

        SELECT 'NO SE PUDO INGRESAR EL ARTICULO' ERROR
```

Guía 4 Página 3 Ejercicio 4.3.1.C

c. Mostrar un mensaje informativo acerca de si hay que reponer o no stock de un artículo cuyo código sea enviado por parámetro

```
create proc pa_23_1c
@cod int
as
declare @difstk int
select @difstk=stock-stock_minimo --se puede guardar descripcion para mostrar
from articulos
where cod_articulo=@cod
if @difstk<=0
       select @cod as codigo,'reponer stock' as mensaje --se puede mostrar la descripcion
else
       select @cod as codigo,'stock ok' as mensaje --se puede mostrar la descripcion

exec  pa_23_1c 1
```

Guía 4 Página 3 Ejercicio 4.3.1.D

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

```
CREATE PROC Actualizar_precio
@precio decimal(10,2),
@porcentaje decimal(10,2)
AS
if @precio<1
BEGIN
SET @precio=@precio*@porcentaje
END
ELSE
SELECT 'No se modifico precio';
```

Guía 4 Página 3 Ejercicio 4.3.1.E

e. Mostrar el nombre del cliente al que se le realizó la primer venta en un parámetro de salida.

```
CREATE PROCEDURE FIRST_CLIENT
@name VARCHAR (50) OUTPUT
AS
SELECT  top 1 @name = C.nom_cliente
FROM FACTURAS F
JOIN CLIENTES C ON C.cod_cliente = F.cod_cliente
order by F.FECHA asc
```

Guía 4 Página 3 Ejercicio 4.3.2.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.

```
create function f_devolverCadena
(@codigo int)
returns varchar(100)
as
begin
        declare @cliente varchar(100)
        set @cliente = (select CONCAT(ape_cliente,' ',nom_cliente,'
',convert(varchar(50),nro_tel),' ',calle,'
',convert(varchar(50),altura),' ',barrio)
                                        from clientes c
                                        join barrios b on c.cod_barrio = b.cod_barrio
                                        where c.cod_cliente = @codigo)
        return @cliente
end


select dbo.f_devolverCadena(2)
```

Guía 4 Página 3 Ejercicio 4.3.2.b

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’).

```
create proc pa_23_2b
@orden varchar(1)='A'
as
       if @orden='A'
              select cod_articulo,descripcion,pre_unitario from articulos
              order by pre_unitario
       else
              select cod_articulo,descripcion,pre_unitario from articulos
              order by pre_unitario desc

exec pa_23_2b 'D'
```

Guía 4 Página 3 Ejercicio 4.3.2.c

c. Crear una función que devuelva el precio al que quedaría un artículo en caso de aplicar un porcentaje de aumento pasado por parámetro.

```
create function f_23_2c
(@cod int,@por decimal(3,1))
returns @pre_art table
(cod int, descrip varchar(100),pre money)
as
begin
	insert @pre_art select cod_articulo,descripcion,pre_unitario*(1+@por/100) from ar-
ticulos
		where cod_articulo=@cod
	return
end

select *
from dbo.f_23_2c(1,10)
```

Guía 4 Página 3 Ejercicio 4.4.1.A

1. Crear un desencadenador para las siguientes acciones:
a. Restar stock DESPUES de INSERTAR una VENTA

```
create trigger dis_ventas_stock
  on detalle_facturas
  after insert
 as
   update articulos set stock = a.stock - inserted.cantidad
   from articulos a
   join inserted on inserted.cod_articulo = a.cod_articulo;
```

Guía 4 Página 3 Ejercicio 4.4.1.B

b. Para no poder modificar el nombre de algún artículo

```
create trigger dis_articulos_actualizar_nombre
  on articulos
  for update
  as
  if update(descripcion)
  begin
  raiserror('No se puede modificar el nombre de un
  artículo', 10, 1)
  rollback transaction
  end;
```

Guía 4 Página 3 Ejercicio 4.4.1.C

c. Insertar en la tabla HistorialPrecio el precio anterior de un artículo si el mismo ha cambiado

```
create trigger precioAnterior
on articulos
for update
as
declare @precioAnterior decimal(10,2), @codigo int, @fecha_desde
DateTime
select @precioAnterior = a.pre_unitario, @codigo = a.cod_articulo from
articulos a
           join inserted
           on a.cod_articulo = inserted.cod_articulo
select top 1 @fecha_desde = fecha_hasta from historial_precios
           where cod_articulo = @codigo
           order by fecha_hasta
           desc
if update(pre_unitario)
begin
insert into historial_precios (cod_articulo, precio, fecha_desde,
fecha_hasta)
values (@codigo, @precioAnterior, @fecha_desde, getdate())
end
else
begin
raiserror('El precio no se ha modificado', 16, 1)
rollback transaction
end
```

Guía 4 Página 3 Ejercicio 4.4.1.D

d. Bloquear al vendedor con código 4 para que no pueda registrar ventas en el sistema.

```
create trigger dis_ventas_vend4
  on facturas
  for insert
 as
if exists (select * from facturas f join inserted i on f.nro_factura = i.nro_factura  where
i.cod_vendedor = 4)
         begin
              print 'vendedor 4 no puede vender'
              --rollback transaction
         end
```

Guía 4 Página 4 Ejercicio 4.5.1

1. Crear un procedimiento almacenado que devuelva la primera y la última venta en una sola tabla

```
create table #temp_pri_ult (
       fecha datetime,
       nro_factura int,
       cod_cliente int,
       cod_vendedor int
       )

create procedure pri_ult_ventas
       as
       insert into #temp_pri_ult
       select top 1 min(f.fecha), nro_factura, cod_cliente, cod_vendedor from facturas f
       group by nro_factura, cod_cliente, cod_vendedor
       insert into #temp_pri_ult
       select top 1 max(f.fecha), nro_factura, cod_cliente, cod_vendedor from facturas f
       group by nro_factura, cod_cliente, cod_vendedor


     exec  pri_ult_ventas


select * from #temp_pri_ult

drop table #temp_pri_ult
```

Guía 4 Página 4 Ejercicio 4.5.2

2. Crear un procedimiento almacenado que devuelva en una sola tabla las facturas realizadas en días impares dentro de un mes / año pasado por parámetro

```
         create table #temp_dias_impar (
       fecha datetime,
       nro_factura int,
       cod_cliente int,
       cod_vendedor int
       )


 create procedure dias_impares_ventas
       @mes int,
       @anio int
       as
       insert into #temp_dias_impar
       SELECT fecha, nro_factura, cod_cliente, cod_vendedor from facturas f
       where  day (f.fecha) % 2 <> 0 and month(f.fecha) = @mes and year(f.fecha) = @anio


     exec  dias_impares_ventas  06, 2022


select * from #temp_dias_impar

drop table #temp_dias_impar
```