BD2 | 2do Parcial

1) La siguiente sentencia:

```
CREATE FUNTION ClientesPorTipoConSaldos(@Tipo tinyint)

returns @T TABLE (idCliente int, NombreCliente nvarchar(100), Saldo money) as ...
```

a) Funcion que devuelve un unico valor dependiendo de los parametros que recibe.
b) Esta función se encarga de devolver todos los clientes por tipo, pero además les aporta el saldo en su cuenta
c) Esta consulta devolverá todos los registros de la tabla Clientes Seleccionados y los registros de la vista ClientesPorTipo para cada tipo de la tabla primaria.

2) Al ejecutar el siguiente código:

```
Declare @DESCRIPCIONERROR VARCHAR(MAX)

Select @DESCRIPCIONERROR = 'Id inexistente'

If exist (select 1 from empleado where id = 2)

Begin

Throw @DESCRIPCION ERROR, 16, 1

End
```

a) Se genera el mensaje de error 'id inexistente' si en la tabla empleado existe el empleado con ID con valor 1

b) Se genera el mensaje de error 'id inexistente' si en la tabla empleado NO existe el empleado con ID con valor 2

c) Se genera el mensaje de error 'id inexistente' si en la tabla empleado solo existe un empleado con ID con valor 2

3) Considerando la siguiente cabecera del procedimiento SQL y los tipos de parámetros, indicar cual es la llamada adecuada

```
CREATE PROCEDURE [DBO].[SP_PROYECTO_ABM]
@ACCION char(1), @ID numeric(18,0), @NOMBRE varchar(100), @FECHAALTA datetime,
@RDO varchar(max) OUTPUT
AS
```

a) EXEC [dbo].[SP_PROYECTO_ABM] ‘A’, 1, ‘P1’, ‘2018/03/12’, @RDO
b) EXEC [dbo].[SP_PROYECTO_ABM] ‘A’, 1, ‘P1’, ‘20180312’, @RDO OUT
c) EXEC [dbo].[SP_PROYECTO_ABM] ‘A’, 1, ‘P1’, ‘20180312’, ‘Resultado’ OUT
d) EXEC [dbo].[SP_PROYECTO_ABM] ‘A’, 1, ‘P1’, ‘20180312’, @RDO OUTPUT

4) Luego que se crea un STORED PROCEDURE

a) La primera vez que se invoca, el motor lo compila y a partir de ahí, se sigue usando la versión compilada del mismo sin importar sus modificaciones.
b) El motor compila cada vez que se invoca
c) La primera vez que se invoca, el motor lo compila y a partir de ahí, se sigue usando la versión compilada del mismo, hasta que se modifique o se reinicie el servidor de SQL

5) Para lograr configurar bajas lógicas sobre los registros de una tabla (y evitar la baja física de los mismos), se deben desarrollar:

a) Triggers del tipo ALTER sobre la operación del DELETE
b) Restricciones del DEFAULTS en el campo que refleja la baja del registro
c) Triggers del tipo INSTEAD OF sobre la operación del DELETE

6) El siguiente código:

```
SELECT @DESCRIPCIONERROR = ‘Nombre invalido’
IF @Nombre is null or RTRIM(LTRIM(@Nombre)) = ‘’
Begin
  Throw @DESCRIPCIONERROR, 16, 1
End
```

a) Valida que la variable @Nombre no sea nula ni sea una cadena vacia
b) Valida solamente que la variable @Nombre no sea nula
c) El código es incorrecto

7) El siguiente código es un Trigger asociado a INSERT:

```
IF EXIST (SELECT 1 FROM DELETED)
   SET @Operation = ‘M’
ELSE
   SET @Operation = ‘A’
```

a) Sirve para diferenciar el tipo de acción que disparo el Trigger
b) Sirve para detectar si la acción se ejecuto correctamente
c) El código es incorrecto

8) ¿Cuál de las siguientes declaraciones de procedimientos almacenados posee un parámetro llamado Codigo de tipo INT opcional con valor predeterminado 0?

a) ALTER PROC TarjetaCredito @Codigo INT DEFAULT 0 AS [..]
b) ALTER PROC TarjetaCredito @Codigo = 0 [..]
c) ALTER PROC TarjetaCredito @Codigo INT = 0 as [..]

9) Escribir una sentencia SQL que cree un procedimiento almacenado llamado EliminarProductos que recibe un parámetro de tipo int. La función de este procedimiento es eliminar de la tabla Production.Product el producto recibido en el parámetro. Programar control de errores usando Try-Catch. Si se produce un error devolverlos siguientes datos del error: Numero, Descripcion, Procedimiento donde se produjo el error y el numero de línea.

```
create or alter procedure EliminarProductos @Id int
as
BEGIN
    begin try
        delete from Production.Product where ProductID = @Id
    end try
    begin catch
        declare @ERROR_NUMBER int = ERROR_NUMBER()
        declare @ERROR_MESSAGE varchar(100) = ERROR_MESSAGE()
        declare @ERROR_PROCEDURE varchar(100) =
ERROR_PROCEDURE()
        declare @ERROR_LINE int = ERROR_LINE()

        select @ERROR_NUMBER as Numero, @ERROR_MESSAGE as
Descripcion, @ERROR_PROCEDURE as Procedimiento, @ERROR_LINE as
Linea
    end catch
END
```

10) Realizar un Trigger que dada la tabla Empleado no permita borrar mas de un registro. El mensaje debe ser ‘NO SE PUEDE BORRAR MAS DE UN EMPLEADO A LA VEZ’.

```
Create or alter trigger tr_UserTable on USER_TABLE
instead of delete
as
begin
    declare @Error varchar(100)
    if (select count(*) from deleted) > 1
    BEGIN
        set @Error = 'NO SE PUEDE BORRAR MAS DE UN EMPLEADO A
LA VEZ'
        select @Error
    END
    else
        delete from USER_TABLE where id in (select id from
deleted)
end
```