Gestión de Datos. Práctica de T-SQL

1. Escriba una función que dados los parámetros código de fabricante y periodo retorne la diferencia de tiempo (en el periodo indicado S:semanas, D: días) entre el promedio de tiempo de embarque de las ordenes que están compuestas solo por productos que él fabrica y el tiempo que el proveedor demora en la entrega de dichos productos.
Si alguno de los parámetros es incoherente se retornara el valor -1

```
Create function ej1 (@manu_code char(3), @periodo char(1))
returns int
BEGIN
declare @resultado int

if @periodo='d' and exists (select * from manufact where manu_code=@manu_code)
begin
      set @resultado=(select avg(datediff(day, o.order_date, o.ship_date))
                        from orders o where  @manu_code = all
                         (select i2.manu_code
                from items i2
                where i2.order_num=o.order_num))-
              (select lead_time from manufact where manu_code=@manu_code)
      return @resultado
end
else if @periodo='s' and exists (select * from manufact where manu_code=@manu_code)
begin
      set @resultado=(select avg(datediff(week, o.order_date, o.ship_date))
            from orders o where  @manu_code =
                all (select i2.manu_code from items i2
                      where i2.order_num=o.order_num))-
                    ((select lead_time from manufact
                       where manu_code=@manu_code)/7)
      return @resultado
end

return -1

END

select dbo.ej1 ('HRO','d')

select dbo.ej1 ('HRO','s')
```

2. Cree una nueva tabla prod_mismo_pedido (stock_num1,manu_code1,stock_num2,manu_code2,cantidad).
   Una vez creada la tabla cree un objeto de base de datos que al ser ejecutado con un parámetro cant_min int, complete la tabla con la lista de productos que hayan sido vendidos juntos en al menos cant_min ocasiones. Se considera que han sido vendidos juntos si se encuentran en la misma orden. El campo cantidad debe ser completado con la cantidad que veces que se vendieron juntos. El ejercicio debe ser resuelto con y sin la utilización de cursores

```
CREATE TABLE prod_mismo_pedido
(stock_num1 SMALLINT,
Manu_code1 CHAR(3),
stock_num2 SMALLINT,
Manu_code2 CHAR(3),
              Cantidad SMALLINT)

create procedure ej2 @cant_min int
as
if (@cant_min <0)
       begin
              RAISERROR('DatoInvalido',14,1)
              return
       end

INSERT INTO prod_mismo_pedido
SELECT i.stock_num prod1,i.manu_code, i2.stock_num prod2,
       i2.manu_code, COUNT(*)
FROM items i JOIN items i2
            ON (i.order_num=i2.order_num and i.stock_num > i2.stock_num)
                                                -- para que los pares no se repitan
GROUP BY i.stock_num, i.manu_code, i2.stock_num, i2.manu_code
HAVING COUNT(*)>=@cant_min



exec dbo.ej2 2
```

3. Cree una nueva tabla Cust_extrm (cust_num, total_orders, total_paid, position).
Una vez creada la tabla cree un objeto de base de datos que al ser ejecutado con un parámetro n int, complete la tabla con los n clientes que más órdenes tengan relacionadas y con los n clientes que menos ordenes tengan relacionadas. Para ambos casos se deberá completar cust_num con el código de cliente , total orders con la cantidad de ordenes asociadas a él, total_paid con la suma de todo lo que haya abonado por las ordenes y ordenes tiene y '-' si pertenece al grupo de los que menos ordenes tiene.
En caso que no existan n clientes la tabla se completara con la totalidad de los clientes tant en el grupo + como en el grupo -.
Si existe más de 1 cliente con la misma cantidad de órdenes se considerara el customer_num para definir cual se inserta preponderando siempre al de menor número
El ejercicio debe ser resuelto con y sin la utilización de cursores

```
Create proc ej3 @n int
AS
if (@n <0)
      begin
            RAISERROR('DatoInvalido',14,1)
            return
      end
insert into Cust_extrm
select top (@n) o.customer_num, COUNT(distinct o.order_num),
       SUM(i.total_price),'+'
from orders o, items i
where  o.order_num=i.order_num
group by o.customer_num
order by COUNT(distinct o.order_num) DESC

insert into Cust_extrm
select top (@n) o.customer_num, COUNT(distinct o.order_num),
SUM(i.total_price),'-'
from orders o, items i
where  o.order_num=i.order_num
group by o.customer_num
order by COUNT(distinct o.order_num)


exec dbo.ej3 2
```

4. Cree un objeto de base de datos que permita recodificar los fabricantes sin alterar la estructura de datos y sin perder información en ningún caso.
Defina los parámetros necesarios del objeto y las validaciones que considere necesarias.

```
create proc ej4 @code_ant char (3), @code_nuevo char(3)
AS
 alter table products
 nocheck constraint FK__stock__manu_code__286302EC;
 alter table items
 nocheck constraint FK__items__25869641;
 alter table catalog
 nocheck constraint FK__catalog__21B6055D;

update manufact
set manu_code=@code_nuevo
where manu_code =@code_ant;

update products
set manu_code=@code_nuevo
where manu_code=@code_ant;

update items
set manu_code=@code_nuevo
where manu_code=@code_ant;

update catalog
set manu_code=@code_nuevo
where manu_code=@code_ant;


alter table items
 check constraint FK__items__25869641;
alter table catalog
 check constraint FK__catalog__21B6055D;
alter table products
 check constraint FK__stock__manu_code__286302EC;


exec dbo.ej4 'ANZ','ANC'
```

5. Cree el/los objetos de base de datos necesarios para que se revise (y modifique los datos de ser necesario) el cumplimiento de una nueva regla de negocio que dice “Por cada 2 unidades que ordene un cliente que correspondan a un mismo fabricante se aplicará un descuento del 2% acumulativo (2 unidades 2%, 4 unidades 4%, etc. Hasta un tope de 20% máximo) en las ordenes del mes siguiente de dicho cliente que tengan al menos 1 producto de ese fabricante”

6. Cree el/los objetos de base de datos necesarios para que automáticamente se cumpla la siguiente regla de negocio sin alterar la estructura de las tablas. La regla a implementar es “Las fechas de la tabla orders deben si o si cumplir con la siguiente regla order_date< ship_date < paid_date. En caso que alguna de las fechas sea null solo se realizaran las verificaciones posibles”. No se conoce como se accede a la tabla orders ni como se manipulan los datos de la misma

```
alter trigger ej6
on orders
for insert, update
as
      if (select count(*) from inserted
            where not (order_date < ship_date and ship_date < paid_date)) > 0
            begin
            RAISERROR('Fecha Inválida', 14, 1)
            rollback
            end
      else
            commit
```

```
INSERT INTO orders (order_num,order_date,customer_num,ship_date,paid_date)
VALUES (1059,'1998/04/11',101,'1998/02/22','1998-06-03')
```

7. Cree una tabla y el/los objetos de base de datos necesarios para almacenar una auditoria detallada de la tabla Stock. En dicha tabla se deben almacenar todos los datos posibles para poder identificar como se llego al estado de cada uno de los registros. ”. No se conoce como se accede a la tabla products ni como se manipulan los datos de la misma

```
CRATE TABLE products_audit
(audit_id INT PRIMARY KEY IDENTITY(1,1),
 stock_num INT,
 manu_code CHAR(3),
 description VARCHAR(50),
 unit_price NUMERIC(8,2),
 unit CHAR(4),
 unit_descr VARCHAR(50),
 operacion CHAR(1),
fechayhora DATETIME DEFAULT GETDATE(),
usuario  VARCHAR(20) DEFAULT USER_NAME()
)
```

```
CREATE TRIGGER tr_products_in
ON products
FOR insert
AS
insert into stock_audit
 SELECT  stock_num,  manu_code,  description,  unit_price,  unit,  unit_descr,  'I',
getdate() ,user_name()
from inserted

CREATE trigger tr_stock_del
ON products
FOR delete
AS
insert into products_audit
 SELECT  stock_num,  manu_code,  description,  unit_price,  unit,  unit_descr,  'D',
getdate() ,user_name()
from DELETED

CREATE trigger tr_stock_up
ON products
FOR UPDATE
AS
insert into products_audit
 SELECT  stock_num,  manu_code,  description,  unit_price,  unit,  unit_descr,  'V',
getdate() ,user_name()
  FROM deleted
insert into stock_audit
SELECT  stock_num,  manu_code,  description,  unit_price,  unit,  unit_descr,  'N',
getdate() ,user_name()
from inserted
```

8. Altere la tabla call_type agregando una columna counter y cree el/los elementos de base de datos necesarios para que en dicha columna se vaya almacenando automáticamente la cantidad de llamados de cada tipo que se realizan.

```
create trigger ej8
   on cust_calls
   for insert
   AS
   declare @customer_num  smallint
   declare @call_dtime     datetime
   declare @call_code      char(1)
         declare c_call cursor
         for select customer_num, call_dtime, call_code from inserted

         open c_call
         fetch  from c_call into @customer_num, @call_dtime, @call_code
         while @@fetch_status=0
               BEGIN
               update call_type
               set cantidad=(1+cantidad)
               where call_code=@call_code
               fetch next from c_call into @customer_num, @call_dtime, @call_code
               END

         close c_call
         deallocate c_call

   commit



   /* prueba inserts individuales */
   insert into cust_calls (customer_num, call_dtime, user_id,
   call_code)values(127,'1998/09/8',1,'D')
   insert into cust_calls (customer_num, call_dtime, user_id,
   call_code)values(126,'1998/09/8',1,'D')


   /* prueba varios inserts*/
   insert into cust_calls select * from pepe


   select * from call_type

   select COUNT(*) from cust_calls group by call_code
```

9. Cree el/los objetos de base de datos necesarios para que automáticamente se cumpla la siguiente regla de negocio sin alterar la estructura de las tablas. La regla a implementar es “Ninguna orden con backlog=y puede tener más de 10 items”. No se conoce como se accede a las tablas orders e items ni como se manipulan los datos de las misma

```
create trigger ej9
on items
instead of insert
AS
BEGIN
  DECLARE @order_num  INT;
  DECLARE @cant_item_inserted INT;
  DECLARE @backlog CHAR(1);

  SELECT @order_num=order_num, @cant_item_inserted = COUNT(*)
  FROM inserted
  GROUP BY order_num;

SELECT @backlog=backlog
FROM orders
WHERE order_num = @order_num

IF @backlog = 'y'
  BEGIN
   IF (SELECT COUNT(*)+@cant_item_inserted FROM items
        WHERE order_num=@order_num)<=10
      BEGIN
           INSERT INTO items
           SELECT * FROM inserted
      END
   ELSE
      BEGIN
         RAISERROR
         ('Error - No puede insertar Ordenes con backlog y mas de 10 items', 14, 1)
         ROLLBACK
      END
  END
ELSE
  BEGIN
       INSERT INTO items
       SELECT * FROM inserted
  END
END
```

```
--Prueba de Inserts
--insert into orders values (9999,getdate(),104,'','y','',NULL,10,10,NULL)

--insert into items values (1,9999,1,'HRO',1,1)
--insert into items values (2,9999,1,'HRO',1,1)
--insert into items values (3,9999,1,'HRO',1,1)
--insert into items values (4,9999,1,'HRO',1,1)
--insert into items values (5,9999,1,'HRO',1,1)
--insert into items values (6,9999,1,'HRO',1,1)
--insert into items values (7,9999,1,'HRO',1,1)
--insert into items values (8,9999,1,'HRO',1,1)
--insert into items values (9,9999,1,'HRO',1,1)
--insert into items values (10,9999,1,'HRO',1,1)

--insert into items values (11,9999,1,'HRO',1,1)
-- Msg 50000, Level 14, State 1, Procedure ej9, Line 28
--Error - No puede insertar Ordenes con backlog y mas de 10 items
--Msg 3609, Level 16, State 1, Line 1
--The transaction ended in the trigger. The batch has been aborted.
```

10. Cree el/los objetos de base de datos necesarios para que ante un intento de borrado de un estado en vez de ejecutarse el borrado físico se modifique su descripción por “Estado baja”. No se conoce como se accede a la tabla State ni como se manipulan los datos de la misma

```
create trigger ej10
on state
instead of delete
AS
begin
update state
set sname= 'Estado baja'
where code in (select code from deleted)
end
```