**Evaluación de Aprendizaje N°3**

Base de datos 2

Alumno: Marcos Cabral

DNI: 42684487

Profesores: Osores Hernán. Pardieux Eliana.

Fecha de Entrega: 2/12/2020.

Para esta evaluación cree las siguientes tablas, que son las que considero necesarias para lo que solicita la evaluación. Además, no se pedía en la evaluación, pero adjunto algunos inserts para las tablas, y scripts para los triggers para comprobarlo más rápido.

create database Eda3Bd2MarcosCabral;

use Eda3Bd2MarcosCabral;

CREATE TABLE Cliente(

codigoCliente int NOT NULL,

nombres varchar(30) not null,

apellido varchar(30) not null,

fecha\_baja date null,

primary key (codigoCliente)

);

CREATE TABLE Libro(

codigoLibro int NOT NULL,

importe float not null default(0),

titulo varchar(30) not null,

stock int not null default (0),

primary key (codigoLibro),

);

CREATE TABLE Prestamo(

numeroPrestamo int NOT NULL,

fechaPrestamo date,

fechaDevolucion date,

codigoCliente int NOT NULL,

primary key(numeroPrestamo),

foreign key (codigoCliente) references Cliente(codigoCliente),

);

CREATE TABLE LibroPrestamo(

numeroPrestamo int NOT NULL,

codigoLibro int NOT NULL,

estadoDevuelto bit,

primary key(numeroPrestamo,codigoLibro),

foreign key (numeroPrestamo) references Prestamo(numeroPrestamo),

foreign key (codigoLibro) references Libro(codigoLibro)

);

--Dejo algunos inserts

insert into Cliente values(1,'Marcos','Cabral',null),(2,'Jorge','Perez',null),

(3,'Martin','Disalvo',null),(4,'Jose','Cruz',null),(5,'Florencia','Lopez',null);

insert into Libro values(1,7000,'Martin Fierro',500),(2,3000,'Caballo de troya 1',100),

(3,2000,'Caballo de troya 2',50),(4,5500,'Harry potter 1',246);

insert into Prestamo values(1,'2020-10-10','2020-10-17',1),(2,'2020-09-19','2020-09-28',2),

(3,'2020-10-01','2020-11-01',3),(4,'2020-10-12','2020-10-30',1);

Punto 1

1)

create table Auditoria(

movimiento int identity not null primary key,

tabla varchar(50) not null,

operacion char(1) not null,

codigo varchar(50) not null,

fecha smalldatetime not null default GETDATE(),

detalle varchar(200),

usuario varchar(50) not null default suser\_sname()

);

2) Asumo que para actualizar el stock se debe insertar o modificar la relación nn entre libro y préstamo, es por esto que trabajo sobre esta tabla.

create trigger tg\_ActualizaStock

on LibroPrestamo

after INSERT, UPDATE

as

begin

declare @numeroPrestamo int

declare @codigoLibro int

declare @estadoDevuelto bit

set @numeroPrestamo = (select numeroPrestamo from inserted)

set @codigoLibro = (select codigoLibro from inserted)

set @estadoDevuelto = (select EstadoDevuelto from inserted)

if (@estadoDevuelto > 0) --si se devuelve

begin

update LibroPrestamo

set EstadoDevuelto = @EstadoDevuelto

where numeroPrestamo = @numeroPrestamo

update Libro

set Stock =(stock-1)

where codigoLibro = @codigoLibro

end

else

begin

update Libro

set Stock = (stock+1)

where codigoLibro = @codigoLibro

end

end;

--script de prueba

select \* from libro where codigoLibro=1;

--el libro 1 tenia 500, ahora 499.

insert into LibroPrestamo (numeroPrestamo,codigoLibro,estadoDevuelto)values(1,1,1)

--devuelvo el libro al terminar el préstamo, seteando el estadoDevuelto a 0.

update LibroPrestamo

set estadoDevuelto=0

where codigoLibro=1 and numeroPrestamo=1;

--se le suma uno al stock ya que lo devolvió, ahora es 500.

select \* from libro where codigoLibro=1;

3)

create trigger tg\_BajaLogicaPrueba

on Cliente

instead of delete

as

begin

declare @id int

declare @fecha smalldatetime

set @id=(select codigoCliente from deleted);

set @fecha=(select fecha\_baja from Cliente where codigoCliente=@id);

if(@fecha is null) --si es null no tiene una fechabaja, entonces le agrego una

begin

update Cliente

set fecha\_baja=GETDATE()

where codigoCliente=@id

insert into Auditoria(tabla,operacion,codigo,detalle)

values('Cliente','B',@id,'Baja logica al cliente');

end

else --si no es null ya tiene una fecha, entonces muestro que ya esta dado de baja

begin

print'El cliente ya fue dado de baja'

rollback transaction

end

end;

--script de prueba, probarlo dos veces con el mismo cliente.

delete from Cliente where codigoCliente=5

4)

create trigger tg\_UpdateImporte

on Libro

instead of update

as

begin

declare @id int

declare @importeViejo int

declare @importeNuevo int

declare @titulo varchar(50)

declare @stock int

set @id = ''+(select codigoLibro from inserted)+''

set @importeNuevo=(select importe from inserted)

set @importeViejo=(select importe from deleted)

set @titulo=(select titulo from inserted)

set @stock=(select stock from inserted)

if(@importeNuevo!=@importeViejo)

begin

print 'Error el importe NO es actualizable'

rollback

end

else

begin

update Libro

set titulo=@titulo,

stock=@stock

where codigoLibro=@id

insert into Auditoria(tabla,operacion,codigo,detalle)

values('Libro','M',@id,'Actualizacion de un libro menos el importe');

end

end;

--script de prueba, el primer update lo debería permitir, el segundo no.

update Libro

set titulo='Harry potter editado'

where codigoLibro=4;

update Libro

set importe=00000

where codigoLibro=4;

5)

create trigger tg\_Empleado\_AM

on Cliente

for insert,update

as

begin

declare @id int

declare @fecha smalldatetime

declare @fechaAnterior smalldatetime

set @id=(select codigoCliente from inserted)

set @fecha=(select fecha\_baja from inserted)

set @fechaAnterior =(select fecha\_baja from deleted)

if(@fecha is not null and @fechaAnterior is not null)

throw 51000,'Tabla Cliente - Error al actualizar datos de auditoría',1;

end

;

--script de prueba, el primer update lo debería permitir ya que cambio el nombre, el segundo no lo permite ya que tiene una fecha de baja.

select \* from Cliente

update Cliente

set nombres='Cambio uno que permite' where codigoCliente=3;

update Cliente

set nombres='Cambio uno que no permite' where codigoCliente=5;

6) Para ejemplificar un deadlock, voy a poner en contexto una situación en la que tenemos dos clientes trabajando en la misma base de datos, al mismo tiempo.

El cliente Marcos, inicia una transacción y realiza un update actualizando el nombre del Libro1, PERO no finaliza la transacción (no ejecuta commit tran).

Por otro lado, tenemos un cliente Germán, que realiza un update actualizando el nombre del Libro2, y tampoco finaliza la transacción.

De momento no hay errores ni deadlock.

Hasta que el cliente Marcos decide actualizar el mismo registro que modificó el cliente Germán (Libro2), y esta consulta se quedará bloqueada.

Al mismo tiempo, el cliente Germán, tiene por casualidad modificar el mismo registro que había modificado en un principio el cliente Marcos (Libro1).

Esto finalmente ocasiona un interbloqueo o deadlock, ya que dos procesos o transacciones se bloquearon entre sí, y el mismo ocurrió ya que un proceso mantuvo un lock sobre un recurso que otro cliente requería.

7) El isolation o nivel de aislamiento que permite acceder a los datos sin efectuar ningún tipo de lockeo es el Read Uncommited, ya que el mismo es el menos restrictivo de los isolations. Esto es por ejemplo, que omite los bloqueos realizados por otras transacciones, o el acceso a lecturas no confirmadas.

8) El isolation o nivel de aislamiento que utiliza SQLServer por defecto es el Read Commited, o Lecturas Confirmadas, es decir que no leen datos sin confirmar, sino que aguarda a que los mismos se encuentren confirmados.

9) Una transacción puede terminar, en rasgos generales, de dos maneras, con éxito o con un error.

Si en algún punto de la transacción ocurre una falla, la misma terminará en su totalidad, esto con el fin de volver al estado inicial en el que se encontraba en la base de datos (rollback tran).

Por otro lado, una transacción puede terminar con éxito, es decir que las operaciones que se ejecutaron fueron completadas de forma correcta, y se debe definir el fin de la misma (commit tran).

10)

create proc DarDeAltaPrestamo @numeroPrestamo int,@fecha date,@codigoCliente int

as

begin

begin tran

begin try

insert into Prestamo(numeroPrestamo,fechaPrestamo,codigoCliente)

values(@numeroPrestamo,@fecha,@codigoCliente);

commit tran

end try

begin catch

SELECT ERROR\_NUMBER() as 'Numero de Error', ERROR\_MESSAGE() as

Mensaje, ERROR\_PROCEDURE() as Procedimiento, ERROR\_LINE() as

Linea, ERROR\_SEVERITY() as Severidad, ERROR\_STATE() as Estado

rollback tran

end catch

end;

--script de prueba

exec DarDeAltaPrestamo 5,'1999-02-06',1