Crear las sentencias SQL que permiten realizar cada ejercicio. La entrega se deberá realizar en PDF con formato "NombresApellido_DNI.PDF"

Triggers

1- Crear una tabla llamada Auditoria que contenga los siguientes campos:

Columna	Tipo de Dato	Observaciones
Movimiento	Int	Identity – No soporta nulos - PK
Tabla	Varchar(50)	No soporta Nulos
Operacion	Char(1)	No soporta Nulos
Codigo	Varchar(50)	No soporta Nulos
Fecha	smalldatetime	No soporta Nulos. Default GetDate()
Detalle	Varchar(200)	
Usuario	Varchar(50)	No soporta Nulos. Default suser_sname()

Aclaración: En el campo Tabla es el nombre de la tabla (ejemplo 'Cliente'), campo operación 'A', 'B','M' (Alta,Baja o Modificación), campo Codigo el código que identifica al registro ingresado, campo Detalle información que considere necesaria.

En todos los casos insertar los datos correspondientes en la tabla de acuerdo con la transacción realizada de alta, baja o modificación.

2- **`tg_ActualizaStock`**: Generar un trigger que actualice el stock del libro cuando se inserte o modifique un préstamo de un libro.

3- **`tg_BajaLogica`**: Cada vez que se borre un cliente, completar la fecha de baja con la fecha actual. Si el cliente, ya posee una fecha de baja, se debe arrojar un error indicando que el cliente ya fue dado de baja.

4- **`tg_UpdateImporte:`** Generar un trigger que arroje un mensaje de error cada vez que se quiera actualizar el importe de un libro indicando que el importe NO es actualizable. El resto de los campos sí se podrían cambiar. Si el importe intenta ser actualizado, la transacción debe invalidarse.

5- tg_Empleado_AM: Generar un trigger que al agregar o modificar un cliente valide si esta dado de baja en ese caso debe mostrar un mensaje 'Tabla Cliente - Error al actualizar datos de auditoría'.

6- Ejemplifique un deadlock
7- ¿Qué isolation permite acceder a los datos sin efectuar ningún tipo de lockeo? Ejemplifique
8- ¿Cuál es el isolation que SQLServer Utiliza por defecto?

9- ¿Cómo puede terminar una transacción?

10- Crear un procedimiento almacenado para dar de alta un préstamo de libros. Debe contener manejo de errores y transacciones. Los datos se reciben por parámetro.

```
USE Biblioteca;

--1--

CREATE TABLE Biblioteca.Auditoria(

Movimiento INT NOT NULL IDENTITY (1,1),
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()

CONSTRAINT PK_Movimiento PRIMARY KEY CLUSTERED (Movimiento)

);


/*2- tg_ActualizaStock: Generar un trigger que actualice el stock del libro cuando se
inserte
o modifique un préstamo de un libro.*/

ALTER TABLE Biblioteca.Libro
ADD stock INT;

UPDATE Biblioteca.Libro SET stock = 10;

CREATE TRIGGER Biblioteca.tg_ActualizaStock
ON Biblioteca.PrestamoLibro
AFTER INSERT,UPDATE
AS
DECLARE @stock INT
DECLARE @idLibro INT
DECLARE @registroLibro INT
DECLARE @registroPrestamo INT
DECLARE @RegistroPrestamoLibro VARCHAR(50)
DECLARE @registroUpdate INT

SET @idLibro = (SELECT codigoLibro FROM inserted)
SET @stock = (SELECT stock FROM Libro WHERE codigoLibro = @idLibro)
SET @registroLibro = (SELECT codigoLibro FROM inserted)
SET @registroPrestamo = (SELECT numeroPrestamo FROM inserted)
SET @registroPrestamoLibro = CONCAT('codigoLibro: ', @registroLibro,' , ',
'numeroPrestamo: ',@registroPrestamo)
```

```
SET @registroUpdate = (SELECT COUNT(*) FROM deleted)

IF(@stock >= 0)
BEGIN
	UPDATE l SET
	L.stock = (L.stock - 1) FROM
	Biblioteca.Libro L
	WHERE @idLibro = l.codigoLibro

	INSERT INTO Biblioteca.Auditoria (Tabla,Operacion,Codigo,Detalle) VALUES
('Libro','M',@registroLibro,'UPDATE stock libro')

	IF(@registroUpdate > 0)
	BEGIN
		INSERT INTO Biblioteca.Auditoria (Tabla,Operacion,Codigo,Detalle) VALUES
('PrestamoLibro','M',@registroPrestamoLibro,'UPDATE prestamo libro')
	END
	ELSE
	BEGIN
		INSERT INTO Biblioteca.Auditoria (Tabla,Operacion,Codigo,Detalle) VALUES
('PrestamoLibro','A',@registroPrestamoLibro,'INSERT prestamo libro')
	END

	COMMIT TRANSACTION
END
ELSE
BEGIN
ROLLBACK TRANSACTION;
THROW 50000,'No hay stock',1
END

-- inserts prestamos y prestamos libro --
INSERT INTO Biblioteca.Prestamo (fechaPrestamo,fechaDevolucion,codigoCliente) VALUES
('2016-10-10','2016-10-10',3),
('2016-10-10','2016-10-10',3),
('2016-10-10','2016-10-10',3),
('2016-10-10','2016-10-10',3),
('2016-10-10','2016-10-10',3),
('2016-10-10','2016-10-10',3),
('2016-10-10','2016-10-10',3),
('2016-10-10','2016-10-10',3)
,('2016-10-10','2016-10-10',3)
,('2016-10-10','2016-10-10',3);

INSERT INTO Biblioteca.PrestamoLibro (numeroPrestamo,codigoLibro) VALUES (19,5);

UPDATE Biblioteca.PrestamoLibro SET codigoLibro = 3 WHERE numeroPrestamo = 19;

/*3- tg_BajaLogica: Cada vez que se borre un cliente, completar la fecha de baja con la
fecha
actual. Si el cliente, ya posee una fecha de baja, se debe arrojar un error indicando
que el cliente
ya fue dado de baja.*/

CREATE TRIGGER Biblioteca.tg_BajaLogica
ON Biblioteca.Cliente
INSTEAD OF DELETE
AS

DECLARE @fechaBaja DATE
DECLARE @codigoClienteDeleted INT

SET @codigoClienteDeleted = (SELECT d.codigoCliente FROM deleted d)
```

```
SET @fechaBaja = (SELECT c.fechaBaja FROM Biblioteca.Cliente c WHERE c.codigoCliente =
@codigoClienteDeleted)

IF(@fechaBaja IS NULL)
BEGIN
	UPDATE Biblioteca.Cliente SET fechaBaja = GETDATE()
	WHERE codigoCliente = @codigoClienteDeleted

	INSERT INTO Biblioteca.Auditoria (Tabla,Operacion,Codigo,Detalle) VALUES
('Cliente','M',@codigoClienteDeleted,'UPDATE fecha baja cliente')

	COMMIT TRANSACTION
END
ELSE
BEGIN
ROLLBACK TRANSACTION;
THROW 50000,'El cliente ya fue dado de baja',1
END

--COMPRUEBO--
DELETE FROM Biblioteca.Cliente WHERE codigoCliente = 4;


/*4- tg_UpdateImporte: Generar un trigger que arroje un mensaje de error cada vez que
se
quiera actualizar el importe de un libro indicando que el importe NO es actualizable.
El resto de
los campos sí se podrían cambiar. Si el importe intenta ser actualizado, la transacción
debe
invalidarse.*/

CREATE TRIGGER Biblioteca.tg_UpdateImporte
ON Biblioteca.Libro
INSTEAD OF UPDATE
AS

DECLARE @importeLibroActual INT
DECLARE @importeLibroNuevo INT
DECLARE @idLibro INT
DECLARE @tituloLibro VARCHAR(255)
DECLARE @codigoEditorial INT
DECLARE @stock INT

SET @tituloLibro = (SELECT tituloLibro FROM inserted)
SET @codigoEditorial = (SELECT codigoEditorial FROM inserted)
SET @stock = (SELECT stock FROM inserted)
SET @idLibro = (SELECT codigoLibro FROM inserted)
SET @importeLibroActual = (SELECT importe FROM deleted WHERE codigoLibro = @idLibrO)
SET @importeLibroNuevo = (SELECT importe FROM inserted)

IF(@importeLibroActual = @importeLibroNuevo)
BEGIN
	UPDATE Biblioteca.Libro SET importe = @importeLibroNuevo,tituloLibro =
@tituloLibro,codigoEditorial = @codigoEditorial,stock = @stock
	WHERE codigoLibro = @idLibro;
	INSERT INTO Biblioteca.Auditoria (Tabla,Operacion,Codigo,Detalle) VALUES
('Libro','M',@idLibro,'Se actulizó libro');
	COMMIT TRANSACTION;
END
ELSE
BEGIN
ROLLBACK TRANSACTION;
THROW 50000,'El importe de un libro no es actualizable',1
```

```sql
END

--COMPRUEBO--

UPDATE Biblioteca.Libro SET tituloLibro = 'titulo libro update 4', stock = 6 ,
codigoEditorial = 2 , importe = 0 WHERE codigoLibro = 1;
UPDATE Biblioteca.Libro SET importe = 1 WHERE codigoLibro = 1;

/*5- tg_Empleado_AM: Generar un trigger que al agregar o modificar un cliente valide si
esta
dado de baja en ese caso debe mostrar un mensaje 'Tabla Cliente - Error al actualizar
datos de
auditoría'. */

CREATE TRIGGER Biblioteca.tg_Empleado_AM
ON Biblioteca.Cliente
INSTEAD OF INSERT,UPDATE
AS

DECLARE @codigoCliente INT
DECLARE @nombreCliente VARCHAR(25)
DECLARE @apellidoCliente VARCHAR(25)
DECLARE @fechaBaja DATE
DECLARE @registroUpdate INT

SET @codigoCliente = (SELECT codigoCliente FROM inserted)
SET @nombreCliente = (SELECT nombreCliente FROM inserted)
SET @apellidoCliente = (SELECT apellidoCliente FROM inserted)
SET @fechaBaja = (SELECT fechaBaja FROM deleted)
SET @registroUpdate = (SELECT COUNT(*) FROM deleted)


IF(@fechaBaja IS NULL)
BEGIN
	IF(@registroUpdate>0)
	BEGIN
		UPDATE Biblioteca.Cliente SET nombreCliente = @nombreCliente WHERE
codigoCliente = @codigoCliente
		UPDATE Biblioteca.Cliente SET apellidoCliente = @apellidoCliente WHERE
codigoCliente = @codigoCliente
		INSERT INTO Biblioteca.Auditoria (Tabla,Operacion,Codigo,Detalle) VALUES
('Cliente','M',@codigoCliente,'UPDATE cliente')
	END
	ELSE
	BEGIN
		INSERT INTO Biblioteca.Cliente(nombreCliente,apellidoCliente,fechaBaja)
VALUES (@nombreCliente,@apellidoCliente,NULL)
		INSERT INTO Biblioteca.Auditoria (Tabla,Operacion,Codigo,Detalle) VALUES
('Cliente','A',@codigoCliente,'INSERT cliente')
	END
	COMMIT TRANSACTION
END
ELSE
BEGIN
ROLLBACK TRANSACTION;
THROW 50000,'Tabla Cliente - Error al actualizar datos de auditoría',1
END

--COMPRUEBO--

INSERT Biblioteca.Cliente(nombreCliente,apellidoCliente,fechaBaja) VALUES
('Ian2','Cepeda2',NULL)
```

```
UPDATE Biblioteca.Cliente SET nombreCliente = 'Diego Armando', apellidoCliente =
'Maradona' where codigoCliente = 4;
```

6- Un deadlock se puede producir cuando dos o más procesos se bloquean entre sí. Esto se da cuando un proceso mantiene un bloqueo sobre un recurso que otro proceso requiere. Un ejemplo se puede dar en el manejo de cuentas bancarias, en el que un usuario A quiere hacer un update al mismo tiempo que el usuario B quiere hacer un update del mismo registro (cuenta). Esto se puede evitar mediante el maneja de excepciones (try/catch) para evitar que haya un “proceso víctima” el cual pierda los datos.

7- El Read Uncommited o lecturas no confirmadas omite los bloqueos realizados por otras transacciones. Las transacciones que se ejecutan con este nivel de aislamiento pueden leer los datos modificados que aún no han confirmado otras transacciones. Esto permite que pueda darse lecturas de datos sucios (datos que no han llegado a validarse), lecturas no repetibles (dos sentencias select iguales y consecutivas que devuelvan datos diferentes), y lecturas fantasma (dos sentencias select iguales y consecutivas en la que puede aparecer o desaparecer registro)

Por otro lado los Snapshots también permite leer los datos sin ningún tipo de bloqueo, leyendo los datos de la transacción omitiendo las modificaciones efectuadas después del inicio de la transacción actual. Esta estrategia permite evitar todos los problemas mencionados en el caso anterior, sin necesidad de bloquear las filas.

8- El Isolation que utiliza SQL Server por defecto es el Read Commited o Lecturas confirmadas.

9- COMMIT TRANSACTION, ROLLBACK TRANSACTION

```
/*10-Crear un procedimiento almacenado para dar de alta un préstamo de libros.
Debe contener manejo de errores y transacciones.
Los datos se reciben por parámetro. */

CREATE PROCEDURE p_AltaPrestamoLibro(@codigoLibro INT,@fechaDevolucion
DATE,@codigoCliente INT)
AS
BEGIN TRANSACTION
DECLARE @numeroPrestamoInsertado INT

IF NOT EXISTS (SELECT * FROM Biblioteca.Libro WHERE codigoLibro = @codigoLibro)
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50000,'El libro no existe',1
	END

IF NOT EXISTS (SELECT * FROM Biblioteca.Cliente WHERE codigoCliente = @codigoCliente)
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50000,'El cliente no existe',1
	END

IF ((SELECT stock FROM Biblioteca.Libro WHERE codigoLibro = @codigoLibro) = 0)
```

```
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50000,'El libro no tiene stock',1
	END

BEGIN TRY
	INSERT INTO Biblioteca.Prestamo(fechaPrestamo,fechaDevolucion,codigoCliente)
VALUES (GETDATE(),@fechaDevolucion,@codigoCliente);
	SET @numeroPrestamoInsertado = (SELECT MAX(numeroPrestamo) FROM
Biblioteca.Prestamo);

		IF(@numeroPrestamoInsertado IS NOT NULL)
			BEGIN
			INSERT INTO Biblioteca.PrestamoLibro(numeroPrestamo,codigoLibro)
VALUES (@numeroPrestamoInsertado,@codigoLibro);
			COMMIT TRANSACTION;
			END
		ELSE
		BEGIN
			ROLLBACK TRANSACTION;
			THROW 50000,'No se pudo setear numero prestamo insert',1
		END
END TRY
BEGIN CATCH

	ROLLBACK TRANSACTION;
	THROW 50000,'Insert en tabla prestamo falló',1

END CATCH


--EXECUTE--

BEGIN
EXECUTE p_AltaPrestamoLibro
@codigoLibro = 1,
@fechaDevolucion = '2016-10-10',
@codigoCliente = 5
END


INSERT INTO Biblioteca.Prestamo(fechaPrestamo,fechaDevolucion,codigoCliente) VALUES
(GETDATE(),'2016-10-10',5);
INSERT INTO Biblioteca.PrestamoLibro(numeroPrestamo,codigoLibro) VALUES(40,5);
```