```sql
go;
IF object_id('dbo.populaContactosDiarios','P') IS NOT NULL
    DROP PROCEDURE dbo.populaContactosDiarios
go;
create proc populaContactosDiarios
as
begin
    SET NOCOUNT ON;
    set transaction isolation level read committed;
    begin transaction
    begin try
        delete from ContactoDiario;
        insert into ContactoDiario(numCliente)
            select top(100) num
            from Cliente
            order by dataUltimoContacto asc;
        commit
    end try
    begin catch
rollback
        raiserror('Erro no populaContactosDiarios',16,1);
    end catch
end
go;
```

Procedures

```sql
public IList<Cliente> obtemClientesAContactar(){

    IList<Cliente> result = new List<Cliente>();
    using(SI2_2021_SV_1epEntities ctx = new SI2_2021_SV_1epEntities()){

        var last100Clientes = ctx.Cliente.OrderBy(clienteAux =>
            clienteAux.dataUltimoContacto).Take(100);

        foreach( Cliente c in last100Clientes){
            result.Add(c);
        }
        Console.WriteLine(String.Format("Clientes, total={0}", result.Count()));
    }
    return result;
    }
}
```

EF

```sql
GO
IF object_id('dbo.vw_summary_intervention','V') IS NOT NULL
    DROP VIEW dbo.vw_summary_intervention
GO
CREATE VIEW dbo.vw_summary_intervention
AS
SELECT i.intervention_code, i.description AS intervention_description, i.state AS intervention_state,
       i.price AS intervention_price,i.start_date AS intervention_start_date, i.end_date,i.asset_id,
       a.brand AS asset_brand, a.acquisition_date AS asset_acquisition_date,
       a.asset_name, a.asset_reference, a.location AS asset_location, a.manager AS asset_manager,
       a.model AS asset_model, a.state AS asset_state, a.type AS asset_type
FROM INTERVENTION i
JOIN ASSET a ON i.asset_id = a.id
```

view

```sql
CREATE TRIGGER removeContactoDiario
ON cliente AFTER UPDATE
AS
BEGIN
    delete from ContactoDiario where numCliente in
        (
            select i.num from INSERTED i
            inner join DELETED d on i.num = d.num
            where i.dataUltimoContacto<>d.dataUltimoContacto
        )
END
```

trigger

```sql
GO
IF OBJECT_ID(N'dbo.trg_updateInterState', N'TR') IS NOT NULL
    DROP TRIGGER dbo.trg_updateInterState;
GO
CREATE TRIGGER trg_updateInterState
ON dbo.vw_summary_intervention
INSTEAD OF UPDATE
AS
BEGIN TRY
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED
    BEGIN TRANSACTION
        DECLARE @state VARCHAR(50)
        DECLARE @intervention INT
        SELECT @state = intervention_state, @intervention = intervention_code FROM INSERTED

        IF UPDATE (intervention_state)
            EXEC dbo.updateStateIntervention @intervention ,@state, NULL
        COMMIT
    END TRY
    BEGIN CATCH
        SELECT ERROR_LINE() AS ErrorLine ,ERROR_MESSAGE() AS ErrorMessage;
        ROLLBACK
END CATCH
```

trigger

```sql
CREATE FUNCTION anosSemcampeao(@anoI int, @anoF int)
RETURNS @ret TABLE (anosSemcampeao int)
AS
BEGIN
    declare @i int = @anoI;
    while @i <= @anoF
    begin
        insert into @ret values(@i);
        set @i = @i+1
    end
    delete from @ret where anoSemCampeao in (
        select ano from campeoes where ano>=@anoI andano<=@anoF
        )

    RETURN
END
```

function

Duas operações num escalonamento S **conflituam** se se verificarem, simultâneamente, as seguintes condições:
1.As operações pertencem a transacções diferentes
2.Ambas as operações acedem ao mesmo item de dados
3.Pelo menos uma das operações é uma operação de escrita

```sql
class DBHelperTS{
    1 reference
    public static string connectionString =
        ConfigurationManager.ConnectionStrings["ex1c"].ConnectionString;
}
0 references
public class ContactosDiariosMapperTS{
    0 references
    public void preencheContactosDiarios(){
        var options = new TransactionOptions();
        options.IsolationLevel = System.Transactions.IsolationLevel.ReadCommitted;

        using (TransactionScope ts = new TransactionScope(TransactionScopeOption.Required,options)){
            using (SqlConnection con = new SqlConnection(DBHelperTS.connectionString)){
                con.Open();
                String sql = "populaContactosDiarios";

                using (SqlCommand cmd = new SqlCommand(sql, con)){
                    cmd.CommandType = CommandType.StoredProcedure;
                    cmd.ExecuteNonQuery();
                }
                ts.Complete();
            }
        }
    }
}
```

ADO.NET

Cascadeless - se nenhuma das suas transacções ler um item escrito por outra transacção ainda não terminada.
Recuperável - se não existir nenhuma transacção que faça commit tendo lido um item depois de ele ter sido escrito por outra transacção ainda não terminada com commit.

Não ser recuperável => não ser "cascadeless"

Estrito - se nenhuma das suas transações ler nem escrever um item escrito por outra transação ainda não terminada.
Série - se as operações de T são executadas consecutivamente
Serializável (do ponto de vista de conflito)?????

```sql
if( not exists (select * from where id= -1))
    insert into equipas(id, descr) values(-1,"**");
insert into campeoes(id, ano, pontos)(select -1,anosSemcampeao,0 from dbo.anosSemcampeao(@anoI,@anoF))
```

outros

Lost update - (conflito **W/W**) Não ocorre com a norma ISO SQL
Dirty read/temporary update- (conflito **W/R**) ("uncommited dependency") Escalonamentos que exibem cascading rollback

Nível de isol.	Anomalia		
	dirty read	nonrep. read	phanton
read uncomm.	sim	sim	sim
read comm.	não	sim	sim
repeat. read	não	não	sim
serializable	não	não	não

O nível read uncommited só é possível com modo read only.
Para se evitarem lost updates, todas as transacções têm de ter o nível de isolamento repeatable read ou superior, ou, então, as actualizações não podem depender de valores resultantes de leituras anteriores (actualizações de uma única acção update).

		Transacção 2		
	Modo	Unlock	Shared	Exclusive
Transacção 1	Unlock	Sim	Sim	Sim
	Shared	Sim	Sim	Não
	Exclusive	Sim	Não	Não

Método	Descrição
ExecuteNonQuery	Permite a execução de comando SQL como INSERT, DELETE, UPDATE e SET
ExecuteScalar	Permite executar comandos que retornem um único valor. Ex. Agregações.
ExecuteReader	Este método permite a execução de comandos que retomam tuplos. Por questões de performance, a execução é feita utilizando o procedimento armazenado `sp_executesql`.

trigger

```
CREATE TRIGGER insCampeaoDefault
ON campeoes INSTEAD OF INSERT
AS
BEGIN
    IF (NOT EXISTS( select id from equipas WHERE id = -1))
    BEGIN
        INSERT INTO equipas(id, descr) VALUES(-1, "**")
    END
INSERT INTO campeoes(id, ano, pontos) select id, ano, pontos from INSERTED;
```

EF

```
public partial class Cliente {
    public Cliente(){
        this.ContactoDiario = new HashSet<ContactoDiario>();
    }
    public int num { get; set; }
    public System.DateTime? dataUltimoContacto { get; set; }
    public virtual ICollection<ContactoDiario> ContactoDiario { get; set; }
}
public partial class ContactoDiario{
        public int id { get; set; }
        public int? numCliente { get; set; }
        public virtual Cliente Cliente { get; set; }
    }
}
```

TRIGGER AFTER	INSERTED contém	DELETED contém
INSERT	Os tuplos inseridos	
UPDATE	Os tuplos atualizados (já com os novos valores)	Os tuplos que foram atualizados, mas com os valores antes da atualização
DELETE		Os tuplos removidos

TRIGGER INSTEAD OF
As tabelas INSERTED e DELETED contêm os tuplos a serem alterados