```
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();
            }
        }
    }
}
```

```
class DBHelper{
    1 reference
    public static string DBString { get =>
        ConfigurationManager.ConnectionStrings["ex4Bd"].ConnectionString; }
}
0 references
class ManutencaoMapper{
    0 references
    public void InsereManutencao(Manutencao m){
        string cs = DBHelper.DBString;
        var options = new TransactionOptions();
        options.IsolationLevel = System.Transactions.IsolationLevel.ReadCommitted;
        using (TransactionScope ts = new TransactionScope(TransactionScopeOption.Required, options)){
            using (SqlConnection con = new SqlConnection(cs)){
                con.Open();
                using (SqlCommand cmd = con.CreateCommand()){
                    if (m.itens.Count == 0)
                        throw new Exception("Manutencao Tem de ter itens a inserir.");
                    addParametersToCommand(cmd);
                    foreach (var item in m.itens){
                        setValuesInCmdParameters(m, item, cmd);
                        executeCommand(cmd);
                    }
                }
            }
            ts.Complete();
        }
    }
    1 reference
    private void addParametersToCommand(SqlCommand cmd){
        cmd.CommandText = "insereManutencaoItem";
        cmd.CommandType = CommandType.StoredProcedure;
        SqlParameter paramMatricula = new SqlParameter("@matricula", SqlDbType.VarChar);
        cmd.Parameters.Add(paramMatricula);
        SqlParameter paramKms = new SqlParameter("@kms", SqlDbType.Int);
        cmd.Parameters.Add(paramKms);
        SqlParameter paramData = new SqlParameter("@data", SqlDbType.Date);
        cmd.Parameters.Add(paramData);
        SqlParameter paramNLinha = new SqlParameter("@nLinha", SqlDbType.Int);
        cmd.Parameters.Add(paramNLinha);
        SqlParameter paramValor = new SqlParameter("@valor", SqlDbType.Decimal);
        cmd.Parameters.Add(paramValor);
    }
    1 reference
    private void setValuesInCmdParameters(Manutencao m, ManutencaoItem item, SqlCommand cmd){
        cmd.Parameters[0].Value = m.matricula;
        cmd.Parameters[1].Value = m.km;
        cmd.Parameters[2].Value = m.data.Date;
        cmd.Parameters[3].Value = item.nLinha;
        cmd.Parameters[4].Value = item.valor;
    }
    1 reference
    private static void executeCommand(SqlCommand cmd){
        cmd.Prepare();
        cmd.ExecuteNonQuery();
    }
}
```

```
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;
  }
}
```

```
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; }
    }
}
```

```
private static void listarDespesas(){
    using (SI2_1819_SV_2epEntities ctx = new SI2_1819_SV_2epEntities()){
        var m = ctx.manutencaos.Include("manutencaoItems")
                .Where(mAux => mAux.km == 10000 && mAux.matricula == "01-01-AB") .FirstOrDefault();
                if (m != null){
                    Decimal total = Decimal.Zero;
                    foreach (var item in m.manutencaoItems){
                        if (item.valor > 20)
                            total += item.valor;
                    }
                    Console..WriteLine(String.Format("Manutenção: mat={0}, km={1}, total={2}", m.matricula, m.km, total));
                }
                else  Console.Out.WriteLine("Sem itens...");
    }
}
```

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`.

```
private static void somarKmsAoVeiculo(){
var options = new TransactionOptions();
options.IsolationLevel =System.Transactions.IsolationLevel.ReadCommitted;
using (TransactionScope ts = new TransactionScope(TransactionScopeOption.Required, options)){
    using (SI2_1819_SV_2epEntities ctx = new SI2_1819_SV_2epEntities()){
        var car = ctx.veiculoes.Find("01-01-AB");
        if (car != null){
            car.kmActuais += 10000;
            ctx.SaveChanges();
            ts.Complete();
        }
        else throw new Exception("Carro não existe...");
    }
}
```

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

```
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
```

```
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
```

```
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;
```

```
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;
```

```
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
```

```
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
```

```
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))
```

```
update manutencao
set valorTotal = dbo.ValorTotalManutencao(@matricula, @km)
where matricula = @matricula AND km = @km
```

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)?????

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

Transação 1 \ Transacção 2	Unlock	Shared	Exclusive
Unlock	Sim	Sim	Sim
Shared	Sim	Sim	Não
Exclusive	Sim	Não	Não

Lost update - (conflito **W/W**); Não ocorre com a norma ISO SQL; duas transações a escrever ao mesmo tempo, uma escreve por cima da outra – overwriting – uncommited data; apenas ficamos com a escrita da ultima transação
Evitar 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).
Dirty read/temporary update- (conflito **W/R**) ("uncommited dependency"); Escalonamentos que exibem cascading rollback; T2 escreve – T1 lê – T2 aborta, isto significa que T1 tem um valor que não é verdade (dirty read)
Nonrepeatable read – T1 R(i=1) – T2 W(i=1) – T2 C - T1 R(i=1)(vou ler valores diferentes do 1º read)
nonrepeatable read pode causar lost updates
Phantom: T1 r(all) – T2 W(i=1) – T1 R(all) -> tem mais um tuplo
Para haver **deadlock**, tem de verificar-se o seguinte:
1. Cada uma das transações tem de ter duas ações de leitura ou escrita
2. Tem de haver ações conflituantes cruzadas, ou seja: as primeiras dessas ações de ambas as transações não conflituam entre si (logo ambas podem deixar o item locked) mas cada uma delas conflitua com as segunda ação da outra transação (logo haverá um bloqueio mútuo).
- Uma das transações é abortada e a outra termina com sucesso
Starvation : Se o esquema de seleção de qual das transações bloqueadas terá acesso ao item for injusto, uma transação pode ficar idefenidamente à espera.

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

Range lock:As BDs geralmente utilizam este mecanismo para optimizar a gestão dos locks efetuados quando estes são em demasia. Por exemplo em vez de colocar 1000 locks pode colocar um range lock que engloba todos esses registos.Outra utilização será a implementação do predicate lock através da utilização de range locks.

Read uncommited
O nível read uncommited só é possível com modo read only.
COPIAR CADERNO => locks O nível read uncommited só é possível com modo read only.

Read uncommited
O nível read uncommited só é possível com modo read only.
COPIAR CADERNO => locks O nível read uncommited só é possível com modo read only.

Read uncommited
O nível read uncommited só é possível com modo read only.
COPIAR CADERNO => locks O nível read uncommited só é possível com modo read only.

Read uncommited
O nível read uncommited só é possível com modo read
COPIAR CADERNO => locks O nível read uncommited
possível com modo read only.

O controlo de **concorrência pessimista** assume que uma T2 vai escrever nos dados lidos por T1 e para se proteger bloqueia os recursos à cabeça causando possíveis problemas de performance.
O controlo de **concorrência otimista** permite ler sem bloquear os recursos obrigando a que quando os dados são escritos, que seja feita uma verificação dos dados para verificar se estes não foram alterados entretanto. Caso os dados tenham sofrido alterações é lançada uma excepção para reagir e ou refrescar os dados da BD, ou esmagar os dados ou abortar a operação. Este controlo obriga ao programador a lidar com este cenário explicitamente.