10-06-2019

DI Casus

Rick van Blitterswijk,  
Christiaan Gal

Inhoudsopgave

[1. Exercise A 2](#_Toc10988141)

[2. Exercise B 3](#_Toc10988142)

[2.1 Constraint 1 3](#_Toc10988143)

[2.2 Constraint 2 5](#_Toc10988144)

[2.3 Constraint 3 7](#_Toc10988145)

[2.4 Constraint 4 9](#_Toc10988146)

[2.5 Constraint 5 12](#_Toc10988147)

[2.6 Constraint 6 14](#_Toc10988148)

[2.7 Constraint 7 16](#_Toc10988149)

[2.8 Constraint 8 17](#_Toc10988150)

[2.9 Constraint 9 19](#_Toc10988151)

[2.10 Constraint 10 22](#_Toc10988152)

[2.11 Constraint 11 25](#_Toc10988153)

[3. Exercise C 28](#_Toc10988154)

[3.1 No problems with non-repeatable reads/phantoms 28](#_Toc10988155)

[3.2 Problem with non-repeatable reads/phantoms 30](#_Toc10988156)

[3.3 Solution 31](#_Toc10988157)

[4. Exercise D 33](#_Toc10988158)

[4.1 Queries with execution plans 33](#_Toc10988159)

[4.2 Which queries to optimize 36](#_Toc10988160)

[4.3 Optimized queries 36](#_Toc10988161)

[5. Exercise E 37](#_Toc10988162)

[6. Exercise F 39](#_Toc10988163)

# 1. Exercise A

/\*===== A =======================================================================\*/

--Drop all foreign keys to implement cascading rules

--Set results to text

select ' alter table ' + TABLE\_NAME + ' drop constraint ' + CONSTRAINT\_NAME from INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS where CONSTRAINT\_TYPE = 'FOREIGN KEY'

alter table emp drop constraint emp\_fk\_grd

alter table emp drop constraint emp\_fk\_dep

alter table srep drop constraint srp\_fk\_emp

alter table memp drop constraint mmp\_fk1\_emp

alter table memp drop constraint mmp\_fk2\_emp

alter table term drop constraint trm\_fk\_emp

alter table dept drop constraint dep\_fk\_emp

alter table offr drop constraint ofr\_fk\_crs

alter table offr drop constraint ofr\_fk\_emp

alter table reg drop constraint reg\_fk\_emp

alter table reg drop constraint reg\_fk\_ofr

alter table hist drop constraint hst\_fk\_emp

alter table hist drop constraint hst\_fk\_dep

--Set results to grid

--Implement foreign keys with cascading rules like in the PDM:

alter table srep

add constraint srep\_fk\_emp foreign key (empno) references emp(empno)

on delete cascade

alter table memp

add constraint mmp\_fk1\_emp foreign key (empno) references emp(empno)

alter table memp

add constraint mmp\_fk2\_emp foreign key (mgr) references emp(empno)

alter table hist

add constraint hist\_fk\_emp foreign key (empno) references emp(empno)

on delete cascade

alter table hist

add constraint hist\_fk\_dept foreign key (deptno) references dept(deptno)

alter table dept

add constraint dept\_fk\_emp foreign key (mgr) references emp(empno)

alter table emp

add constraint emp\_fk1\_dept foreign key (deptno) references dept(deptno)

alter table emp

add constraint emp\_fk2\_grd foreign key (sgrade) references grd(grade)

on update cascade

alter table offr

add constraint offr\_fk1\_emp foreign key (trainer) references emp(empno)

on delete set null

alter table offr

add constraint offr\_fk2\_crs foreign key (course) references crs(code)

on update cascade

alter table reg

add constraint reg\_fk1\_offr foreign key (course, starts) references offr(course, starts)

alter table reg

add constraint reg\_fk2\_emp foreign key (stud) references emp(empno)

on delete cascade

alter table term

add constraint term\_fk\_emp foreign key (empno) references emp(empno)

# 2. Exercise B

## 2.1 Constraint 1

/\*

Constraint 1. The president of the company earns more than $10.000 monthly.

Assumption: This constraint does not include Bonuses as this is about monthly earnings.

\*/

/\*======== IMPLEMENTATION =========================================================================================================================================================================================================================\*/

go

CREATE PROCEDURE usp\_updateMsal

--DROP PROCEDURE usp\_updateMsal

(

@empno numeric(4,0),

@newMsal numeric(7,2)

)

AS

BEGIN

declare @tr\_name varchar(10) = 'none'

BEGIN TRY

if @@trancount > 0

begin

set @tr\_name = 'yep'

save tran @tr\_name

end

else

begin

begin tran

end

--if empno corresponds to the president

if((select e.job from emp e where e.empno = @empno) = 'PRESIDENT')

begin

--if msal is greater than 9999

if(@newMsal >= 10000)

begin

update emp

set msal = @newMsal

where empno = @empno

end

--msal is too low

else

THROW 50001, 'President msal can not be less than 10000', 1;

end

--empno does not correspond to the president

else

begin

update emp

set msal = @newMsal

where empno = @empno

end

if @tr\_name = 'none'

COMMIT TRAN

END TRY

BEGIN CATCH

if @tr\_name = 'none'

ROLLBACK TRAN

else

rollback tran @tr\_name

DECLARE @Message nvarchar(2048) = ERROR\_MESSAGE()

raiserror (@Message, 16, 1)

END CATCH

END

go

/\* =================== TESTS =====================================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint1';

--EXEC tSQLt.DropClass 'Constraint1';

GO

CREATE PROCEDURE [Constraint1].[test = 1: update mSal > 10000 on a president]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.ExpectNoException

insert into emp values (1000, 'Hans', 'PRESIDENT', '1957-12-22', '1992-01-01', 11, 11000, 'HANS', 10)

EXEC usp\_updateMsal @empno = 1000, @newMsal = 12000

END

GO

GO

CREATE PROCEDURE [Constraint1].[test = 2: update mSal < 10000 on a president]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.ExpectException 'President msal can not be less than 10000'

insert into emp values (1000, 'Hans', 'PRESIDENT', '1957-12-22', '1992-01-01', 11, 11000, 'HANS', 10)

EXEC usp\_updateMsal @empno = 1000, @newMsal = 9000

END

GO

GO

CREATE PROCEDURE [Constraint1].[test = 3: update msal < 10000 on a manager]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.ExpectNoException

insert into emp values (1000, 'Hans', 'MANAGER', '1957-12-22', '1992-01-01', 11, 11000, 'HANS', 10)

EXEC usp\_updateMsal @empno = 1000, @newMsal = 9000

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint1'

## 2.2 Constraint 2

/\*

Constraint 2. A department that employs the president or a manager should also employ at least one administrator.

On an insert in emp this constraint can be violated in this way:

A manager/president is inserted in with a detpno that has no administrator.

On an update in emp this constraint can be violated in multiple ways:

The job of an ADMIN row is changed but the dept with the same deptno as the deptno in that row has a MANAGER/PRESIDENT.

The job of a ADMIN/OTHER row is changed to MANAGER/PRESIDENT but after this update the dept with the same deptno has no ADMIN

On a delete in emp this constraint can be violated in multiple ways:

The only ADMIN from a certain dept gets deleted but this dept has a MANAGER/PRESIDENT

We will make a delete trigger to ensure this constraint is not violated

\*/

/\*====== IMPLEMENTATION ==============================================================================================================================================================================================================\*/

go

CREATE TRIGGER utr\_deleteEmp

--DROP TRIGGER utr\_deleteEmp

on emp

after delete

AS

BEGIN TRY

--If there is a deleted admin

if (EXISTS (select 1 from deleted where job = 'ADMIN'))

begin

--If there is a dept with a deleted admin that has no more admins

if (not exists (select 1 from emp where deptno in (select deptno from deleted where job = 'ADMIN') and job = 'ADMIN'))

begin

--If the dept from a deleted admin still has a president or manager

if (exists (select 1 from emp where deptno in (select deptno from deleted where job = 'ADMIN') and (job = 'PRESIDENT' or job = 'MANAGER')))

THROW 50002, 'You cant delete an admin from a department with no more admins and a president/manager', 1;

end

end

END TRY

BEGIN CATCH

;THROW

END CATCH

go

/\*====== TESTS ==============================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint2';

--EXEC tSQLt.DropClass 'Constraint2';

GO

CREATE PROCEDURE [Constraint2].[test = 1: Delete the last admin from a dept with a president/manager still there (multiple inserts in 1 statement)]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp'

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.emp', @triggername = 'utr\_deleteEmp'

EXEC tSQLt.ExpectException 'You cant delete an admin from a department with no more admins and a president/manager'

--Make an admin in dept 99

insert into emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)

values(9999,'test','ADMIN','01-24-1969','01-05-1997',3,2900,'MONIQUE1',99);

--Make an admin in dept 98

insert into emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)

values(9997,'test','SALESREP','01-24-1969','01-05-1997',3,2900,'MONIQUE1',98);

--Make a manager in dept 99

insert into emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)

values(9998,'test','MANAGER','01-24-1969','01-05-1997',3,2900,'MONIQUE2',99);

--Delete the admin from dept 99 with manager still there

delete from emp where empno = 9999

delete from emp where empno = 9997

END

GO

CREATE PROCEDURE [Constraint2].[test = 2: Delete the last admin from a dept without a president/manager (multiple inserts in 1 statement)]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp'

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.emp', @triggername = 'utr\_deleteEmp'

EXEC tSQLt.ExpectNoException

--Make an admin in dept 99

insert into emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)

values(9999,'test','ADMIN','01-24-1969','01-05-1997',3,2900,'MONIQUE1',99);

--Make an admin in dept 98

insert into emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)

values(9998,'test','ADMIN','01-24-1969','01-05-1997',3,2900,'MONIQUE1',99);

--Make an admin in dept 98

insert into emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)

values(9997,'test','SALESREP','01-24-1969','01-05-1997',3,2900,'MONIQUE1',98);

--Delete the admin from dept 99 (without a manager)

delete from emp where empno = 9999

delete from emp where empno = 9997

END

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint2'

## 2.3 Constraint 3

/\*

Constraint 3. The company hires adult personnel only.

This constraint can be protected with a declarative implementation.

\*/

/\*======== IMPLEMENTATION =========================================================================================================================================================================================================================\*/

alter table emp add constraint emp\_chk\_born check (DATEDIFF(year, born, GETDATE()) >= 18);

/\*====== TESTS ==============================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint3';

--EXEC tSQLt.DropClass 'Constraint3';

GO

CREATE PROCEDURE [Constraint3].[test = 1: Insert an emp who is a minor]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp';

alter table emp

add constraint test\_emp\_chk\_born check (DATEDIFF(year, born, GETDATE()) >= 18);

EXEC tSQLt.ExpectException 'The INSERT statement conflicted with the CHECK constraint "test\_emp\_chk\_born". The conflict occurred in database "COURSE", table "dbo.emp", column ''born''.'

insert into emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)

values(9999,'test','ADMIN','01-24-2018','01-05-1997',3,2900,'MONIQUE1',10);

END

GO

GO

CREATE PROCEDURE [Constraint3].[test = 2: Insert an emp who is an adult]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp';

alter table emp

add constraint test\_emp\_chk\_born check (DATEDIFF(year, born, GETDATE()) >= 18);

EXEC tSQLt.ExpectNoException

insert into emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)

values(9999,'test','ADMIN','01-24-2000','01-05-1997',3,2900,'MONIQUE1',10);

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint3'

## 2.4 Constraint 4

/\*

4. A salary grade overlaps with at most one lower salary grade. The llimit of a salary grade must be higher than the llimit of the next lower salary grade.

The ulimit of the salary grade must be higher than the ulimit of the next lower salary grade.

We will make an insert trigger to ensure this constraint is not violated

\*/

/\*====== IMPLEMENTATION ==============================================================================================================================================================================================================\*/

go

CREATE TRIGGER utr\_insertGrd

--DROP TRIGGER utr\_insertGrd

on grd

after insert

AS

BEGIN TRY

--If the llimit of the next lower salary grade is higher than the llimit of the inserted grd

if (exists (select 1 from grd g inner join inserted i on g.grade+2 = i.grade where i.llimit < g.llimit))

THROW 50009, 'You cant insert a grade with a llimit lower than that of the next lower grade', 1;

--If the ulimit of the next lower salary grade is higher than the ulimit of the inserted grd

if (exists (select 1 from grd g inner join inserted i on g.grade+2 = i.grade where i.ulimit < g.ulimit))

THROW 50010, 'You cant insert a grade with a ulimit lower than that of the next lower grade', 1;

--If the llimit of the inserted grd is higher than that of the next higher grade

if (exists (select 1 from grd g inner join inserted i on g.grade = i.grade+2 where g.llimit < i.llimit))

THROW 50011, 'You cant insert a grade with a llimit higher than the next higher grade', 1;

--If the ulimit of the inserted grd is higher than that of the next higher grade

if (exists (select 1 from grd g inner join inserted i on g.grade = i.grade+2 where g.ulimit < i.ulimit))

THROW 50012, 'You cant insert a grade with a ulimit higher than the next higher grade', 1;

END TRY

BEGIN CATCH

;THROW

END CATCH

go

/\*====== TESTS ==============================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint4';

--EXEC tSQLt.DropClass 'Constraint4';

GO

CREATE PROCEDURE [Constraint4].[test = 1: Insert a grade with a llimit lower than that of the next lower grade]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.grd'

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.grd', @triggername = 'utr\_insertGrd'

EXEC tSQLt.ExpectException 'You cant insert a grade with a llimit lower than that of the next lower grade'

--Insert first 2 grades

begin tran

insert into grd values(1, 10, 20, 0)

insert into grd values(2, 20, 30, 0)

commit tran

--Insert 3rd grade that will violate the constraint

insert into grd values(3, 5, 40, 0)

END

GO

GO

CREATE PROCEDURE [Constraint4].[test = 2: Insert a grade with a ulimit lower than that of the next lower grade]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.grd'

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.grd', @triggername = 'utr\_insertGrd'

EXEC tSQLt.ExpectException 'You cant insert a grade with a ulimit lower than that of the next lower grade'

--Insert first 2 grades

begin tran

insert into grd values(1, 10, 20, 0)

insert into grd values(2, 20, 30, 0)

commit tran

--Insert 3rd grade that will violate the constraint

insert into grd values(3, 40, 10, 0)

END

GO

GO

CREATE PROCEDURE [Constraint4].[test = 3: Insert a grade with a llimit higher than the next higher grade]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.grd'

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.grd', @triggername = 'utr\_insertGrd'

EXEC tSQLt.ExpectException 'You cant insert a grade with a llimit higher than the next higher grade'

--Insert first 2 grades

begin tran

insert into grd values(5, 10, 20, 0)

insert into grd values(6, 20, 30, 0)

commit tran

--Insert 3rd grade that will violate the constraint

insert into grd values(4, 30, 10, 0)

END

GO

GO

CREATE PROCEDURE [Constraint4].[test = 4: Insert a grade with a ulimit higher than the next higher grade]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.grd'

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.grd', @triggername = 'utr\_insertGrd'

EXEC tSQLt.ExpectException 'You cant insert a grade with a ulimit higher than the next higher grade'

--Insert first 2 grades

begin tran

insert into grd values(5, 10, 20, 0)

insert into grd values(6, 20, 30, 0)

commit tran

--Insert 3rd grade that will violate the constraint

insert into grd values(4, 5, 60, 0)

END

GO

GO

CREATE PROCEDURE [Constraint4].[test = 5: Succesful insert]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.grd'

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.grd', @triggername = 'utr\_insertGrd'

EXEC tSQLt.ExpectNoException

--Insert first 2 grades

begin tran

insert into grd values(1, 10, 20, 0)

insert into grd values(2, 20, 30, 0)

insert into grd values(4, 40, 50, 0)

insert into grd values(5, 50, 60, 0)

commit tran

--Insert 3rd grade that will violate the constraint

insert into grd values(3, 30, 40, 0)

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint4'

## 2.5 Constraint 5

/\* 5. The start date and known trainer uniquely identify course offerings.

Note: the use of a filtered index is not allowed.

the current unique constraint prevents inserting duplicates on the columns trainer and starts.

but it also prevents duplicates of null values in the trainer column with the same date.

this is not the favourable solution as a null value means it can be updated in the future or the trainer of this set course hasnt been decided yet.

therefore we will be dropping the current unique constraint and add a trigger which will check for duplicates (except null values in trainer) on every record.

violating this constraint is still possible when a record is updated to be on the same date with the same trainer.

But we chose a trigger after insert since inserts are more likely to happen.

\*/

/\*====== IMPLEMENTATION ==============================================================================================================================================================================================================\*/

go

CREATE TRIGGER utr\_insertOffr

--DROP TRIGGER utr\_insertOffr

on offr

after insert

AS

BEGIN TRY

if ((select count(\*) from offr o, inserted i where o.starts = i.starts and o.trainer = i.trainer) > 1)

THROW 50003, 'One of the inserted records has the same start date and trainer id as an already existing record. These values must be unique', 1;

END TRY

BEGIN CATCH

;THROW

END CATCH

go

/\*====== TESTS ==============================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint5';

--EXEC tSQLt.DropClass 'Constraint5';

GO

CREATE PROCEDURE [Constraint5].[test = 1: Insert record with already existing start date and trainer id combination.]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.offr', @triggername = 'utr\_insertOffr'

EXEC tSQLt.ExpectException 'One of the inserted records has the same start date and trainer id as an already existing record. These values must be unique'

insert into offr(course,starts,[status],maxcap,trainer,loc)

values('AM4DPM', '1997-09-06', 'CONF', 6, 1017, 'SAN FRANCISCO');

insert into offr(course,starts,[status],maxcap,trainer,loc)

values('AM5DPM', '1997-09-06', 'CONF', 6, 1017, 'SAN FRANCISCO');

END

GO

GO

CREATE PROCEDURE [Constraint5].[test = 2 records on the same date but with null values for trainer (not possible with the previous unique constraint).]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.offr', @triggername = 'utr\_insertOffr'

EXEC tSQLt.ExpectNoException

insert into offr(course,starts,[status],maxcap,trainer,loc)

values('APEX', '1997-09-06', 'CONF', 6, null, 'SAN FRANCISCO');

insert into offr(course,starts,[status],maxcap,trainer,loc)

values('AM4DPM', '1997-09-06', 'CONF', 6, null, 'SAN FRANCISCO');

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint5'

## 2.6 Constraint 6

/\* 6. Trainers cannot teach different courses simultaneously.

on an update or insert in offr table can violate this procedure:

if a trainer is updated/inserted to be giving a course while also giving another course

but the most logic thing to do in our opinion is to create a procedure for inserts as this will happen more frequently.

\*/

/\*====== IMPLEMENTATION ==============================================================================================================================================================================================================\*/

go

CREATE PROCEDURE usp\_insertTrainer

--drop procedure usp\_insertTrainer

(

@course varchar(6),

@starts date,

@status varchar(4),

@maxcap numeric(2),

@trainer numeric(4),

@loc varchar(14)

)

AS

BEGIN

declare @tr\_name varchar(10) = 'none'

BEGIN TRY

if @@trancount > 0

begin

set @tr\_name = 'yep'

save tran @tr\_name

end

else

begin

begin tran

end

declare @startdate date

set @startdate = (select top 1 starts from offr where trainer = @trainer and course = @course and @starts >= starts order by starts asc)

if(@starts >= @startdate and @starts <= DATEADD(day, (select dur from crs where code = @course), @startdate))

THROW 50004, 'the inserted course starts before all courses of this trainer are over. Record cant be inserted.', 1;

else

insert into offr values(@course, @starts, @status, @maxcap, @trainer, @loc)

if @tr\_name = 'none'

COMMIT TRAN

END TRY

BEGIN CATCH

if @tr\_name = 'none'

ROLLBACK TRAN

else

rollback tran @tr\_name

DECLARE @Message nvarchar(2048) = ERROR\_MESSAGE()

raiserror (@Message, 16, 1)

END CATCH

END

go

/\*====== TESTS ==============================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint6';

--EXEC tSQLt.DropClass 'Constraint6';

GO

CREATE PROCEDURE [Constraint6].[test = 1: insert course when another course of the same trainer is not finished yet]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC tSQLt.FakeTable 'dbo.crs';

EXEC tSQLt.ExpectException 'the inserted course starts before all courses of this trainer are over. Record cant be inserted.'

insert into crs values ('AM4DP', 'Applied Math for DB-Pros', 'DSG', 10)

insert into offr values ('AM4DP', '2018-12-30', 'CONF', 6, 1017, 'SAN FRANCISCO')

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2018-12-31', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'SAN FRANCISCO'

END

GO

GO

CREATE PROCEDURE [Constraint6].[test = 2: insert course when all other courses of this trainer are finished]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC tSQLt.FakeTable 'dbo.crs';

EXEC tSQLt.ExpectNoException

insert into crs values ('AM4DP', 'Applied Math for DB-Pros', 'DSG', 1)

insert into offr values ('AM4DP', '2000-12-30', 'CONF', 6, 1017, 'SAN FRANCISCO')

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2018-12-31', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'SAN FRANCISCO'

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint6'

## 2.7 Constraint 7

/\* 7. An active employee cannot be managed by a terminated employee.

On an insert in term this constraint can be violated in this way:

Employee is inserted in term but not deleted in memp. So employees managed by the terminated employee are still in the memp table.

On an update in term this constraint can be violated in this way:

Terminated employee in term is updated to another empno, then data in the memp table will not correspond.

We have decided to create a trigger after an insert on the term table to ensure all employees managed by the inserted manager will be deleted from the memp table.

\*/

/\*======== IMPLEMENTATION =========================================================================================================================================================================================================================\*/

go

CREATE TRIGGER utr\_insertTerm

--drop trigger utr\_insertTerm

on term

after insert

AS

BEGIN TRY

delete from memp where mgr in (select empno from inserted)

END TRY

BEGIN CATCH

;THROW

END CATCH

go

/\* =================== TESTS =====================================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint7';

--EXEC tSQLt.DropClass 'Constraint7';

GO

CREATE PROCEDURE [Constraint7].[test = 1: insert a record in term will delete all records in memp which have that exact empno.]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.term';

EXEC tSQLt.FakeTable 'dbo.memp';

EXEC tSQLt.ExpectNoException

insert into term(empno, leftcomp, comments)

values(1003, '2019-04-05', '');

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint7'

## 2.8 Constraint 8

/\* 8. A trainer cannot register for a course offering taught by him- or herself.

On an insert in reg this constraint can be violated in this way:

The empno is from a teacher that also teaches this course

On an update in reg this constraint can be violated in this way:

The empno is from a teacher that also teaches this course

We have chosen to create a stored procedure on the reg table to ensure new registrations do not allow to register for a course taught by the same employee.

\*/

/\*======== IMPLEMENTATION =========================================================================================================================================================================================================================\*/

go

CREATE PROCEDURE usp\_insertReg

--drop procedure usp\_insertReg

(

@stud numeric(4),

@course varchar(6),

@starts date,

@eval numeric(1)

)

AS

BEGIN

declare @tr\_name varchar(10) = 'none'

BEGIN TRY

if @@trancount > 0

begin

set @tr\_name = 'yep'

save tran @tr\_name

end

else

begin

begin tran

end

if(exists(select 1 from offr where course = @course and trainer = @stud))

THROW 50005, 'the inserted student also teaches this course, this is not allowed.', 1;

else

insert into reg values (@stud, @course, @starts, @eval)

if @tr\_name = 'none'

COMMIT TRAN

END TRY

BEGIN CATCH

if @tr\_name = 'none'

ROLLBACK TRAN

else

rollback tran @tr\_name

DECLARE @Message nvarchar(2048) = ERROR\_MESSAGE()

raiserror (@Message, 16, 1)

END CATCH

END

go

/\* =================== TESTS =====================================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint8';

--EXEC tSQLt.DropClass 'Constraint8';

GO

CREATE PROCEDURE [Constraint8].[test = 1: insert student who also teaches the course]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.reg';

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC tSQLt.ExpectException 'the inserted student also teaches this course, this is not allowed.'

insert into offr values('AM4DPM', '2005-04-03', 'CONF', 6, 1001, 'SAN FRANCISCO')

exec usp\_insertReg @stud = 1001, @course = 'AM4DPM', @starts = '2005-04-03', @eval = 4

END

GO

GO

CREATE PROCEDURE [Constraint8].[test = 2: insert student who does not teach this course]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.reg';

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC tSQLt.ExpectNoException

insert into offr values('AM4DPM', '2005-04-03', 'CONF', 6, 1000, 'SAN FRANCISCO')

exec usp\_insertReg @stud = 1013, @course = 'AM4DP', @starts = '1997-09-06', @eval = 4

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint8'

## 2.9 Constraint 9

/\* 9. At least half of the course offerings (measured by duration) taught by a trainer must be ‘home based’.

Note: ‘Home based’ means the course is offered at the same location where the employee is employed.

On an insert in offr this constraint can be violated in this way:

if half of the courses offered by the trainer are not home based after the insert

On an update in offr this constraint can be violated in this way:

if half of the courses offered by the trainer are not home based after an update on a course

On a delete in offr this constraint can be violated in this way:

if half of the courses offered by the trainer are not home based after the delete

we have chosen to update the already existing stored procedure of inserting new offerings to also check for this constraint.

\*/

/\*======== IMPLEMENTATION =========================================================================================================================================================================================================================\*/

go

CREATE PROCEDURE usp\_insertTrainer

--drop proc usp\_insertTrainer

(

@course varchar(6),

@starts date,

@status varchar(4),

@maxcap numeric(2),

@trainer numeric(4),

@loc varchar(14)

)

AS

BEGIN

declare @tr\_name varchar(10) = 'none'

BEGIN TRY

if @@trancount > 0

begin

set @tr\_name = 'yep'

save tran @tr\_name

end

else

begin

begin tran

end

declare @trainerloc varchar(14)

set @trainerloc = (select loc from dept where deptno in (select deptno from emp where empno = @trainer))

if((select count(\*) from offr where loc = @trainerloc and trainer = @trainer)+1 <= (select count(\*) from offr where trainer = @trainer)/2)

THROW 50006, 'the inserted course should be home-based (same location as the trainer). Else more than half of the courses taught by this trainer are not home-based, this is not allowed.', 1;

else

begin

insert into offr values(@course, @starts, @status, @maxcap, @trainer, @loc)

end

if @tr\_name = 'none'

COMMIT TRAN

END TRY

BEGIN CATCH

if @tr\_name = 'none'

ROLLBACK TRAN

else

rollback tran @tr\_name

DECLARE @Message nvarchar(2048) = ERROR\_MESSAGE()

raiserror (@Message, 16, 1)

END CATCH

END

go

/\* =================== TESTS =====================================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint9';

--EXEC tSQLt.DropClass 'Constraint9';

GO

CREATE PROCEDURE [Constraint9].[test = 1: insert non home-based courses that exceed the max of the constraint (half of total courses)]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC tSQLt.FakeTable 'dbo.dept';

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.ExpectException 'the inserted course should be home-based (same location as the trainer). Else more than half of the courses taught by this trainer are not home-based, this is not allowed.'

insert into emp values (1017, 'Hans', 'PRESIDENT', '1957-12-22', '1992-01-01', 11, 11000, 'HANS', 10)

insert into dept values (10, 'HEAD OFFICE', 'DALLAS', 1001)

exec usp\_insertTrainer @course = 'AM4DP', @starts = '1998-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '1999-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2000-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2001-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2003-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2005-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2006-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2007-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2008-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2007-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

END

GO

GO

CREATE PROCEDURE [Constraint9].[test = 2: insert non home-based course that does not exceed the max of the constraint]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC tSQLt.FakeTable 'dbo.dept';

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.ExpectNoException

insert into emp values (1017, 'Hans', 'PRESIDENT', '1957-12-22', '1992-01-01', 11, 11000, 'HANS', 10)

insert into dept values (10, 'HEAD OFFICE', 'DALLAS', 1001)

exec usp\_insertTrainer @course = 'AM4DP', @starts = '2007-09-17', @status = 'CONF', @maxcap = 6, @trainer = 1017, @loc = 'AMSTERDAM'

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint9'

## 2.10 Constraint 10

/\* 10. Offerings with 6 or more registrations must have status confirmed.

On an insert in reg this constraint can be violated in this way:

if a new student is registered and it now totals to 6 for this course the offr table is not updated to have the status CONF for this course

On an update in reg this constraint can be violated in this way:

if a student is updated to follow another course and it now totals to 6 or more registrations the offr table is not updated to have the status CONF for this course

On an update in offr this constraint can be violated in this way:

if a course is updated to be scheduled even though there are 6 or more students registered for the course

we have chosen to update the already existing stored procedure of inserting new registrations to also check for this constraint.

\*/

/\*======== IMPLEMENTATION =========================================================================================================================================================================================================================\*/

go

CREATE PROCEDURE usp\_insertOffr

--DROP PROCEDURE usp\_insertOffr

(

@stud numeric(4),

@course varchar(6),

@starts date,

@eval numeric(1)

)

AS

BEGIN

declare @tr\_name varchar(10) = 'none'

BEGIN TRY

if @@trancount > 0

begin

set @tr\_name = 'yep'

save tran @tr\_name

end

else

begin

begin tran

end

INSERT INTO reg values (@stud, @course, @starts, @eval)

if((select count(\*) from reg where course = @course and starts = @starts) >= 6)

update offr set status = 'CONF' where course = @course and starts = @starts

if @tr\_name = 'none'

COMMIT TRAN

END TRY

BEGIN CATCH

if @tr\_name = 'none'

ROLLBACK TRAN

else

rollback tran @tr\_name

DECLARE @Message nvarchar(2048) = ERROR\_MESSAGE()

raiserror (@Message, 16, 1)

END CATCH

END

go

/\* =================== TESTS =====================================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint10';

--EXEC tSQLt.DropClass 'Constraint10';

GO

CREATE PROCEDURE [Constraint10].[test = 1: Insert enough students to fill a course (6 total)]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.reg';

EXEC tSQLt.FakeTable 'dbo.offr';

insert into offr values('AM4DP', '2006-08-03', 'SCHD', 6, 1001, 'SAN FRANCISCO')

exec usp\_insertOffr @stud = 1029, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

exec usp\_insertOffr @stud = 1030, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

exec usp\_insertOffr @stud = 1031, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

exec usp\_insertOffr @stud = 1032, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

exec usp\_insertOffr @stud = 1033, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

exec usp\_insertOffr @stud = 1034, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

exec usp\_insertOffr @stud = 1035, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

Declare @actual varchar(40)

set @actual = (select status from offr where course = 'AM4DP')

Declare @expected varchar(40)

set @expected = 'CONF'

EXEC tSQLt.assertEquals @expected, @actual;

END

GO

GO

CREATE PROCEDURE [Constraint10].[test = 2: Insert not enough students to fill a course (2 total)]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.reg';

EXEC tSQLt.FakeTable 'dbo.offr';

insert into offr values('AM4DP', '2005-04-03', 'SCHD', 6, 1001, 'SAN FRANCISCO')

exec usp\_insertOffr @stud = 1029, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

exec usp\_insertOffr @stud = 1030, @course = 'AM4DP', @starts = '2006-08-03', @eval = -1

Declare @actual varchar(40)

set @actual = (select status from offr where course = 'AM4DP')

Declare @expected varchar(40)

set @expected = 'SCHD'

EXEC tSQLt.assertEquals @expected, @actual;

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint10'

## 2.11 Constraint 11

/\* 11. You are allowed to teach a course only if:

your job type is trainer and

- you have been employed for at least one year

- or you have attended the course yourself (as participant)

\*/

/\*======== IMPLEMENTATION =========================================================================================================================================================================================================================\*/

go

CREATE TRIGGER utr\_checkNewTeacher

--drop trigger utr\_checkNewTeacher

on offr

after insert

AS

BEGIN TRY

-- If there is an inserted course that isnt given by a teacher

if(exists(select 1 from inserted i inner join emp e on i.trainer = e.empno where e.job != 'TRAINER'))

THROW 50007, 'the inserted course is not given by a trainer', 1;

-- If there is an inserted course that is given by a teacher that doesnt have a course that started more than a year ago or this teacher hasnt taught any courses yet.

if((not exists(select 1 from inserted where starts in (select i.starts from inserted i inner join offr o on i.trainer = o.trainer where DATEDIFF(year, i.starts, o.starts) >= 1)))

or (not exists (select o.trainer from inserted i inner join offr o on i.trainer = o.trainer)))

begin

-- If that teacher has not attended this course himself

if(not exists(select 1 from inserted i where i.trainer in (select stud from reg)

and i.trainer = (select trainer from inserted where starts in (select i.starts from inserted i inner join offr o on i.trainer = o.trainer where DATEDIFF(year, i.starts, o.starts) < 1))))

THROW 50008, 'the employee of this inserted course has not been an employee for at least one year or has not followed the course himself/herself', 1;

end

END TRY

BEGIN CATCH

;THROW

END CATCH

go

/\* =================== TESTS =====================================================================================================================================================================================================================\*/

EXEC tSQLt.NewTestClass 'Constraint11';

--EXEC tSQLt.DropClass 'Constraint11';

GO

CREATE PROCEDURE [Constraint11].[test = 1: Insert course not given by a teacher]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.offr', @triggername = 'utr\_checkNewTeacher'

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.ExpectException 'the inserted course is not given by a trainer'

insert into emp values (1000, 'Hans', 'PRESIDENT', '1957-12-22', '1992-01-01', 11, 11000, 'HANS', 10)

insert into offr values('AM4DPM', '1997-09-06', 'CONF', 6, 1000, 'SAN FRANCISCO');

END

GO

GO

CREATE PROCEDURE [Constraint11].[test = 2: Insert course not given by a trainer who has been working for more than a year]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.offr', @triggername = 'utr\_checkNewTeacher'

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.ExpectException 'the employee of this inserted course has not been an employee for at least one year or has not followed the course himself/herself'

insert into emp values (9999, 'Hans', 'TRAINER', '1957-12-22', '2019-02-02', 11, 11000, 'HANS', 10)

insert into offr values('AM4DPM', '1997-09-06', 'CONF', 6, 9999, 'SAN FRANCISCO');

insert into offr values('AM4DPM', '1997-09-06', 'CONF', 6, 1000, 'SAN FRANCISCO');

END

GO

GO

CREATE PROCEDURE [Constraint11].[test = 3: Insert course not given by a trainer who has attended the course himself]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.offr', @triggername = 'utr\_checkNewTeacher'

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.FakeTable 'dbo.reg';

EXEC tSQLt.ExpectException 'the employee of this inserted course has not been an employee for at least one year or has not followed the course himself/herself'

insert into reg values (1001, 'AM4DPM','2005-04-03', 4)

insert into emp values (1000, 'Hans', 'TRAINER', '1957-12-22', '2019-02-02', 11, 11000, 'HANS', 10)

insert into offr values('AM4DPM', '1997-09-06', 'CONF', 6, 1000, 'SAN FRANCISCO');

END

GO

GO

CREATE PROCEDURE [Constraint11].[test = 4: Succesful insert]

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr';

EXEC [tSQLt].[ApplyTrigger] @tablename = 'dbo.offr', @triggername = 'utr\_checkNewTeacher'

EXEC tSQLt.FakeTable 'dbo.emp';

EXEC tSQLt.FakeTable 'dbo.reg';

EXEC tSQLt.ExpectNoException

insert into reg values (1000, 'AM4DPM','1-04-03', 4)

insert into emp values (1000, 'Hans', 'TRAINER', '1957-12-22', '1992-01-01', 11, 11000, 'HANS', 10)

insert into offr values('AM4DPM', '1997-09-06', 'CONF', 6, 1000, 'SAN FRANCISCO');

END

GO

/\* ====== EXECUTION ========================================================================================================================================================================================================================================\*/

EXEC [tSQLt].[Run] 'Constraint11'

# 3. Exercise C

## 3.1 No problems with non-repeatable reads/phantoms

|  |  |
| --- | --- |
| **Connection 1 – Read commited** | **Connection 2 – Read commited** |
| BEGIN  declare @tr\_name varchar(10) = 'none'  BEGIN TRY  if @@trancount > 0  begin  set @tr\_name = 'yep'  save tran @tr\_name  end  else  begin  begin tran  end  if((select e.job from emp e where e.empno = 1) = 'PRESIDENT')  Connection selects the job from employee number 1. It sees that that job is ‘PRESIDENT’ so the value in the if statement is true. S-lock was held while the connection was reading the data, but it is now removed after this statement finished. |  |
|  | BEGIN  declare @tr\_name varchar(10) = 'none'  BEGIN TRY  if @@trancount > 0  begin  set @tr\_name = 'yep'  save tran @tr\_name  end  else  begin  begin tran  end  if((select e.job from emp e where e.empno = 1) = 'PRESIDENT')  Connection selects the job from employee number 1. It sees that that job is ‘PRESIDENT’. Connection 1 has not changed this value so there is no nonrepeatable read. Also there are no records in the result of this select statement that weren’t there before the begin transaction of connection 2 was called, so there are no phantom records.The value in the if statement is true. S-lock was held while the connection was reading the data, but it is now removed after this statement finished. |
| Begin  if(1000000 >= 10000)  begin  update emp  set msal = @newMsal  where empno = @empno  Connection 1 puts an S-lock on this data. There is no S-lock on it because connection 2 has already dropped it’s lock because the isolation level is ‘Read commited’.  end  else  THROW 50001, 'President msal can not be less than 10000', 1;  end  else  begin  update emp  set msal = @newMsal  where empno = @empno  end  if @tr\_name = 'none'  COMMIT TRAN  The transaction is commited by connection 1 so all s-locks and x-locks placed by connection 1 are dropped.  END TRY  BEGIN CATCH  if @tr\_name = 'none'  ROLLBACK TRAN  else  rollback tran @tr\_name  DECLARE @Message nvarchar(2048) = ERROR\_MESSAGE()  raiserror (@Message, 16, 1)  END CATCH  END |  |
|  | Begin  if(1000000 >= 10000)  begin  update emp  set msal = @newMsal  where empno = @empno  Connection 2 puts an S-lock on this data. There were no locks on it when this statement was executed.  end  else  THROW 50001, 'President msal can not be less than 10000', 1;  end  else  begin  update emp  set msal = @newMsal  where empno = @empno  end  if @tr\_name = 'none'  COMMIT TRAN  The transaction is commited by connection 2.  END TRY  BEGIN CATCH  if @tr\_name = 'none'  ROLLBACK TRAN  else  rollback tran @tr\_name  DECLARE @Message nvarchar(2048) = ERROR\_MESSAGE()  raiserror (@Message, 16, 1)  END CATCH  END |

## 3.2 Problem with non-repeatable reads/phantoms

|  |  |
| --- | --- |
| **Connection 1 – Read commited** | **Connection 2 – Read commited** |
| go  CREATE TRIGGER utr\_deleteEmp  on emp  after delete  AS  BEGIN TRY  if ((select job from deleted) = 'ADMIN')  If statement is true  begin  if (not exists (select \* from emp where deptno = (select deptno from deleted) and (job = 'ADMIN')))  This if statement is true. Which means there are no more admins in the dept of the deleted admin. S-locks are placed and immediately removed after the select is done. |  |
|  | update emp  set job = 'MANAGER'  where empno = 1  Connection 2 puts an x-lock on empno 1 (which is the same empno as in connection 1) and releases it due to autocommit. |
| begin  if (exists (select \* from emp where deptno = (select deptno from deleted) and (job = 'PRESIDENT' or job = 'MANAGER')))  This if statement is true. So there is a manager or president in this dept. S-locks are placed and immediately removed.  THROW 50002, 'You cant delete an admin from a department with no more admins and a president/manager', 1;  Connection one throws this error that says you can’t delete this admin because it is in a dept with no more admins and it has a president/admin in it. This is false because the emp that is being deleted (empno = 1) is not an admin anymore. Connection 2 changed it to a Manager which means this error is false.  end  end  END TRY  BEGIN CATCH  ;THROW  END CATCH  go |  |

## 3.3 Solution

|  |  |
| --- | --- |
| **Connection 1 – Repeatable read** | **Connection 2 – Read commited** |
| go  CREATE TRIGGER utr\_deleteEmp  on emp  after delete  AS  BEGIN TRY  if ((select job from deleted) = 'ADMIN')  If statement is true  begin  if (not exists (select \* from emp where deptno = (select deptno from deleted) and (job = 'ADMIN')))  This if statement is true. Which means there are no more admins in the dept of the deleted admin. S-locks are placed and held until transaction commit due to isolation level repeatable read. |  |
|  | update emp  set job = 'MANAGER'  where empno = 1  Connection 2 tries to put an x-lock the row with empno = 1 but can’t because connection 1 has put on s-lock on it, this statement can’t be executed. |
| begin  if (exists (select \* from emp where deptno = (select deptno from deleted) and (job = 'PRESIDENT' or job = 'MANAGER')))  This if statement is true. So there is a manager or president in this dept. S-locks are placed and held until transaction commit  THROW 50002, 'You cant delete an admin from a department with no more admins and a president/manager', 1;  Connection one throws this error that says you can’t delete this admin because it is in a dept with no more admins and it has a president/admin in it. This is correct.  end  end  END TRY  BEGIN CATCH  ;THROW  END CATCH  Go After rollback transaction/tansaction commit, S-locks are removed. |  |

# 4. Exercise D

## 4.1 Queries with execution plans

|  |
| --- |
| **Constraint 1** |
| select e.job from emp e where e.empno = @empno |
| https://i.gyazo.com/095346892945f9517a23531c59d7828a.png |

|  |
| --- |
| **Constraint 2** |
| select 1 from deleted where job = 'ADMIN')  select 1 from emp where deptno in (select deptno from deleted where job = 'ADMIN') and job = 'ADMIN'  select 1 from emp where deptno in (select deptno from deleted where job = 'ADMIN') and (job = 'PRESIDENT' or job = 'MANAGER') |
| https://i.gyazo.com/c93f7dc5962ecdad1755180b165653be.png  https://i.gyazo.com/aea1cefb512f34b4bfdfdf7fb493ce01.png  https://i.gyazo.com/e3b1a1d297ee42dfb259eb9137082512.png |

|  |
| --- |
| **Constraint 3** |
| No select queries in this constraint |

|  |
| --- |
| **Constraint 4** |
| select 1 from grd g inner join inserted i on g.grade+2 = i.grade where i.llimit < g.llimit) |
| https://i.gyazo.com/da54fdba3be3a69094503bc4e112bae9.png |

|  |
| --- |
| **Constraint 5** |
| select count(\*) from offr o, inserted i where o.starts = i.starts and o.trainer = i.trainer |
| https://i.gyazo.com/f4e29fea073c6a5ac61ac1620e1d9ecb.png |

|  |
| --- |
| **Constraint 6** |
| select top 1 starts from offr where trainer = @trainer and course = @course and @starts >= starts order by starts asc |
| https://i.gyazo.com/b6d0fe83bf0c1460cc4189638da4e292.png |

|  |
| --- |
| **Constraint 7** |
| select empno from inserted |
| https://i.gyazo.com/840808724ae9df096820b77653d85e3b.png |

|  |
| --- |
| **Constraint 8** |
| select 1 from offr where course = @course and trainer = @stud |
| https://i.gyazo.com/0952526a9d9a13dba9d24d0cff59a142.png |

|  |
| --- |
| **Constraint 9** |
| select loc from dept where deptno in (select deptno from emp where empno = @trainer  select count(\*) from offr where loc = @trainerloc and trainer = @trainer)+1 <= (select count(\*) from offr where trainer = @trainer) |
| https://i.gyazo.com/38da132bad415e2575d8e9a6c05d5df7.png  https://i.gyazo.com/fd099fe8454134e99d5b8d06fcb3cb7c.png |

|  |
| --- |
| **Constraint 10** |
| select count(\*) from reg where course = @course and starts = @starts |
| https://i.gyazo.com/8dac0f4ea7cbf19eaa6541b67430a70c.png |

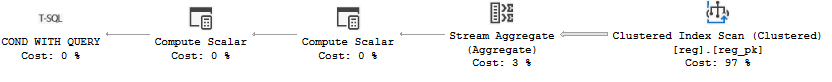
|  |
| --- |
| **Constraint 11** |
| select 1 from inserted i inner join emp e on i.trainer = e.empno where e.job != 'TRAINER' |
| https://i.gyazo.com/c10d322319d72e53503497954291e28e.png |

## 4.2 Which queries to optimize

After executing the queries above and analysing it’s the execution plans. We have examined which 2 queries would benefit performance wise the most of optimization by adding indexes. We used the following criteria:

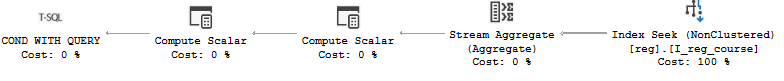
1. Is there a scan?
2. Are there already indexes and how are these used?

## 4.3 Optimized queries

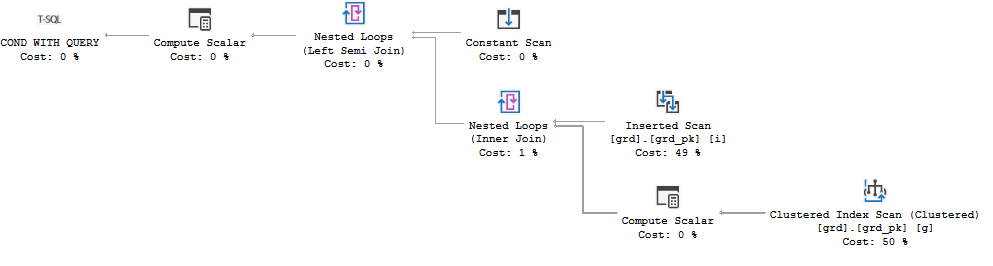
**Constraint 10:**

We chose this constraint because multiple clustered index scans are being performed. Scans are expensive for the performance because they will scan through the entire table (emp table in this case). With indexes a seek can be performed which will increase performance drastically.

Create Index I\_reg\_course on reg(course)

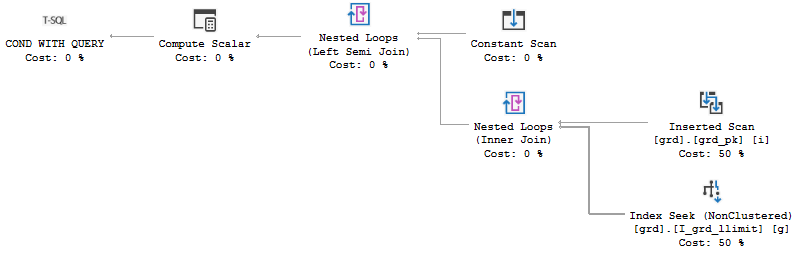


**Constraint 4:**



We chose this constraint for the same reason as constraint 10 and got the same kind of improvements in performance.

Create Index I\_grd\_llimit on grd(llimit)



# 5. Exercise E

CREATE PROCEDURE usp\_create\_table\_with\_triggers

@table varchar(255)

AS

SET NOCOUNT ON

SET XACT\_ABORT OFF

DECLARE @TranCounter INT;

SET @TranCounter = @@TRANCOUNT;

IF @TranCounter > 0

SAVE TRANSACTION ProcedureSave;

ELSE

BEGIN TRANSACTION;

BEGIN TRY

--create an exact copy of the table without the constraints

declare @sql nvarchar(MAX) = ''

set @sql = 'SELECT top 0 \* INTO HIST\_'+@table+' FROM '+@table

exec sp\_sqlexec @sql

--add the timestamp column

set @sql = 'ALTER TABLE HIST\_'+@table+' ADD [timestamp] datetime default getdate() not null'

exec sp\_sqlexec @sql

--get the primary key columns

declare @columns nvarchar(max) = ''

SELECT @columns = @columns + COLUMN\_NAME+','

FROM INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE

WHERE OBJECTPROPERTY(OBJECT\_ID(CONSTRAINT\_SCHEMA + '.' + QUOTENAME(CONSTRAINT\_NAME)), 'IsPrimaryKey') = 1

AND TABLE\_NAME = @table

--Make the timestamp column the primary key

set @columns = @columns + 'timestamp'

set @sql = 'ALTER TABLE HIST\_'+@table+' ADD PRIMARY KEY ('+@columns+');'

exec sp\_sqlexec @sql

--see columns to transfer

declare @Histcolumns nvarchar(max) =''

SELECT @Histcolumns = @Histcolumns+ COLUMN\_NAME+', '

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = @table

set @Histcolumns = SUBSTRING(@Histcolumns,0,LEN(@Histcolumns))

--Create update trigger on the table

set @sql ='create trigger update\_hist\_'+@table+' on '+@table+' for update as

set nocount on

insert into HIST\_'+@table+' ('+@Histcolumns+') select '+@Histcolumns+' from deleted;'

exec sp\_sqlexec @sql

set @sql = 'exec sp\_settriggerorder @triggername = '''+concat('update\_hist\_',@table)+''', @order = ''last'', @stmttype = ''update'''

exec sp\_sqlexec @sql

--Create the delete trigger on the

set @sql ='create trigger delete\_hist\_'+@table+' on '+@table+' for delete as

set nocount on

insert into HIST\_'+@table+' ('+@Histcolumns+') select '+@Histcolumns+' from deleted;'

exec sp\_sqlexec @sql

set @sql = 'exec sp\_settriggerorder @triggername = '''+concat('delete\_hist\_',@table)+''', @order = ''last'', @stmttype = ''delete'''

exec sp\_sqlexec @sql

IF @TranCounter = 0 AND XACT\_STATE() = 1

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @TranCounter = 0

BEGIN

IF XACT\_STATE() = 1 ROLLBACK TRANSACTION;

END;

ELSE

BEGIN

IF XACT\_STATE() <> -1 ROLLBACK TRANSACTION ProcedureSave;

END;

THROW

END CATCH

GO

CREATE PROCEDURE usp\_create\_the\_hist\_tables

AS

SET NOCOUNT ON

SET XACT\_ABORT OFF

DECLARE @TranCounter INT;

SET @TranCounter = @@TRANCOUNT;

IF @TranCounter > 0

SAVE TRANSACTION ProcedureSave;

ELSE

BEGIN TRANSACTION;

BEGIN TRY

declare @sql nvarchar(max) = ''

select @sql = @sql +'exec usp\_createe\_table\_with\_triggers ' +i.[TABLE\_NAME]+';' from INFORMATION\_SCHEMA.TABLES i

where 'HIST\_'+i.TABLE\_NAME not in (select TABLE\_NAME from INFORMATION\_SCHEMA.TABLES)

and i.[TABLE\_NAME] not like 'HIST\_%';

exec sp\_sqlexec @sql

IF @TranCounter = 0 AND XACT\_STATE() = 1

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @TranCounter = 0

BEGIN

IF XACT\_STATE() = 1 ROLLBACK TRANSACTION;

END;

ELSE

BEGIN

IF XACT\_STATE() <> -1 ROLLBACK TRANSACTION ProcedureSave;

END;

THROW

END CATCH

GO

exec usp\_create\_the\_hist\_tables

# 6. Exercise F

/\*\*\*\*\*\* EMPLOYEE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* ================ IMPLEMENTATION ================ \*/

go

use master

go

--Create an employee login

CREATE LOGIN employee1 WITH PASSWORD = 'employee1'

--drop login employee1

--Create the user employee1 on login employee1

go

use COURSE

go

CREATE USER employee1 FROM LOGIN employee1

--drop user employee1

CREATE ROLE employee

--drop role employee

-- Grant the permissions to the employee role

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.reg to employee

GRANT SELECT on dbo.emp TO employee

GRANT SELECT on dbo.offr TO employee

--Make the user employee1 member of the employee role

ALTER ROLE employee ADD MEMBER employee1

/\* ================ TESTS ================ \*/

--Impersonate the employee1 login

EXECUTE AS LOGIN = 'employee1'

--Check which user and loggin you are impersonating

SELECT user as [database user], system\_user as [current login], original\_login() as [originele login]

--The employee user has full acces to the reg table so it can insert, update delete and select data as long as no constraints are violated

begin tran

--Insert a new record

insert into reg values(1000, 'AM4DP', '2001-11-03', 4)

--New record is inserted

select \* from reg

--update the record

update reg

set eval = 3

where course = 'AM4DP' and starts = '2001-11-03' and stud = 1000

--New record updated

select \* from reg

--Delete the record

Delete from reg

where course = 'AM4DP' and starts = '2001-11-03' and stud = 1000

--New record deleted

select \* from reg

rollback tran

--The employee can read data from the EMP and OFFR tables

select \* from emp

select \* from offr

--CRUD actions on other tables are not allowed

select \* from grd

insert into crs values('ADP', 'Applied Math for DB-Pros', 'DSG', 9)

revert

/\*\*\*\*\*\*SERVICE ACCOUNT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* ================ IMPLEMENTATION ================ \*/

go

use master

go

--Create an employee login

CREATE LOGIN serviceaccount WITH PASSWORD = 'serviceaccount'

--drop login serviceaccount

--Create the user employee1 on login employee1

go

use COURSE

go

CREATE USER serviceaccount FROM LOGIN serviceaccount

--drop user serviceaccount

CREATE ROLE serviceaccountrole

--drop role serviceaccountrole

-- Grant the permissions to the employee role

grant select on schema::dbo to serviceaccountrole

--Make the user employee1 member of the employee role

ALTER ROLE serviceaccountrole ADD MEMBER serviceaccount

/\* ================ TESTS ================ \*/

--Impersonate the employee1 login

EXECUTE AS LOGIN = 'serviceaccount'

--Check which user and loggin you are impersonating

SELECT user as [database user], system\_user as [current login], original\_login() as [originele login]

--The Serviceaccount can read from all tables

select \* from crs

select \* from dept

select \* from emp

select \* from grd

select \* from memp

select \* from offr

select \* from reg

select \* from srep

select \* from term

--Inserts, updates and deletes are not allowed

insert into srep values(1011, 10, 100)

delete from crs where code = 'AM4DP'

update grd

set llimit = 10

where grade = 1

revert