|  |
| --- |
| HAN Arnhem |
| DI Casus |
| Course offering database |

|  |
| --- |
| Jannick Joosten (598696) & Erik Knaake (598368)  29 maart 2019  ITA-ISE R-B-f  Course DI  Leo van den Berge  Versie 1.0 |

Inhoudsopgave

[1 Introductie 3](#_Toc4764615)

[1.1 Mappen structuur 3](#_Toc4764616)

[1.2 Installatie database 3](#_Toc4764617)

[2 Opdracht A 4](#_Toc4764618)

[3 Opdracht B 5](#_Toc4764619)

[3.1 Aanpak 5](#_Toc4764620)

[3.2 Test opbouw 5](#_Toc4764621)

[3.3 Constraint 1 7](#_Toc4764622)

[3.3.1 Test 7](#_Toc4764623)

[3.4 Constraint 2 7](#_Toc4764624)

[3.4.1 Test 9](#_Toc4764625)

[3.5 Constraint 3 10](#_Toc4764626)

[3.5.1 Test 10](#_Toc4764627)

[3.6 Constraint 4 11](#_Toc4764628)

[3.6.1 Test 12](#_Toc4764629)

[3.7 Constraint 5 12](#_Toc4764630)

[3.7.1 Test 13](#_Toc4764631)

[3.8 Constraint 6 14](#_Toc4764632)

[3.8.1 Test 15](#_Toc4764633)

[3.9 Constraint 7 16](#_Toc4764634)

[3.9.1 Test 17](#_Toc4764635)

[3.10 Constraint 8 17](#_Toc4764636)

[3.10.1 Test 19](#_Toc4764637)

[3.11 Constraint 9 19](#_Toc4764638)

[3.11.1 Test 20](#_Toc4764639)

[3.12 Constraint 10 21](#_Toc4764640)

[3.12.1 Test 22](#_Toc4764641)

[3.13 Constraint 11 22](#_Toc4764642)

[4 Opdracht C 26](#_Toc4764643)

[4.1 Constraint 5: Geen concurrency problemen 26](#_Toc4764644)

[4.2 Constraint 11: Non Repeatable Read 27](#_Toc4764645)

[5 Opdracht D 30](#_Toc4764646)

[6 Opdracht E 34](#_Toc4764647)

[7 Opdracht F 38](#_Toc4764648)

[8 Literatuurlijst 40](#_Toc4764649)

# Introductie

Voor het vak DI hebben wij de course offering database (HAN, z.d.) moeten implementeren. Dit is een database die gaat over de verschillende courses die het bedrijf geeft, de medewerkers en hun salaris schalen. In deze database kunnen medewerkers zich registreren voor course offerings die door andere medewerkers worden gegeven. Om deze database te implementeren hebben wij verschillende opdrachten gekregen, A t/m F die in dit verslag toegelicht zullen worden. Deze opdracht behandeld alle aspecten die in de course DI naar voren zijn gekomen.

In de volgende hoofdstukken zal een toelichting gegeven worden van onze projectindeling en hoe onze uitwerking te instaleren is.

## Mappen structuur

In de hoofdfolder bevindt zich een bestand genaamd: Di Casus.ssmssln. Wanneer dit bestand geopend wordt door middel van Microsoft SQL Server Management Studio (ssms) zal de folder genaamd: DI Casus geopend worden binnen de Solution Explorer van ssms. Zo hoeft deze folder zelf niet benaderd te worden, echter is in deze folder wel het Word bestand te vinden voor opdracht C die binnen de Solution Explorer niet geopend kan worden.

Verder is er een folder aangemaakt voor alle execution plans. De execution plans die hier in staan zijn als aanvulling op opdracht D.

## Installatie database

Om deze database volledig te installeren zullen een hoop bestanden uitgevoerd moeten worden. Dit moet op een bepaalde volgorde gedaan worden zodat er niets verkeerd kan gaan.

Om het gemakkelijk te houden kan, zoals eerder benoemd, het bestand Di Casus.ssmssln geopend worden met ssms. Vervolgens kan het Solution Explorer venster geopend worden om alle bestanden te zien. Als deze niet aanwezig is in de huidige ontwikkelomgeving kan benaderd worden via: View > Solution Explorer.

Hieronder volgt de volgorde waarin de bestanden uitgevoerd moeten worden:

1. COURSE\_cretab.sql
2. COURSE\_database\_state.sql
3. COURSE\_constraints.sql
4. Opdracht A FK.sql
5. Opdracht B constraints.sql
   1. Om deze constraints te testen, kan de file ‘Constraints tests.sql’ uitgevoerd worden. Hiervoor is echter wel tSQLt nodig. Dit zal zelf nog op de database geïnstalleerd moeten worden.
6. Opdracht D indexing.sql
7. Opdracht E code generatie.sql
8. Opdracht F secutiy.sql

# Opdracht A

Opdracht A luidt: “*Implement the database according to the PowerDesigner PDM you find on the last page of this document. Once again you’ll find the scripts to get you started on OnderwijsOnline. Add to the constraint script the foreign key and the cascading rule declarations as depicted in the PowerDesigner PDM*.” (HAN, z.d.).

In het PDM staan de foreign keys vermeld, met telkens de acties die worden genomen bij een update of delete. Hierbij staat N voor SET NULL, C voor CASCADE en niets voor NO ACTION.

Het volgende stuk code is representatief voor alle foreign keys die wij hebben geïmplementeerd:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FK hist -> emp (ON DELETE CASCADE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALTER TABLE hist

ADD CONSTRAINT FK\_hist\_emp

FOREIGN KEY (empno) REFERENCES emp(empno)

ON UPDATE NO ACTION

ON DELETE CASCADE

go

# Opdracht B

Opdracht B luidt: “*Implement constraints 1 through 11. In case a declarative implementation is possible, provide it.*

*Motivate your choice for stored procedures or triggers. Build at least 3 stored procedures and 3 triggers. If your choices result in less than 3 of each, create alternative solutions to in the end deliver at minimum 3 of each. Analyze what actions on which tables may cause the constraint to be violated (describe all possible scenarios). Implement the in your opinion most logical of the possibly many scenarios that may cause the constraint to be violated.*

***FIRST*** *create the test cases you’ll need test your solution. Add tests with* ***multiple*** *rows if your solution is a trigger (this means at least tests with at minimum two allowed, two dis-allowed and a combination of rows). Only than start implementing the constraint.*

*So, every trigger should correctly handle multiple row SQL statements.  
Minimize the use of variables. Use only one SQL-statement for a condition where ever possible.*

*Use correct error handling and transaction management in both Triggers and Stored Procedures like taught in the course.”* (HAN, z.d.).

In dit hoofdstuk wordt er een toelichting gegeven over de implementatie van de 11 constraints binnen de course database. Om dit document niet te groot te maken, is slechts een klein gedeelte van de test cases opgenomen bij elke constraint.

## Aanpak

Om deze opdracht te realiseren hebben wij telkens eerst een analyse gedaan waaruit blijkt onder welke omstandigheden de constraint fout kan gaan. Hierna hebben we een keuze gemaakt of we een stored procedure, trigger of check constraint gingen gebruiken.

Vervolgens hebben we met tSQLt testcases gebouwd, zodat er eenvoudig te valideren is of de constraint goed werkt. Als laatste hebben we de constraint geïmplementeerd.

## Test opbouw

Elke test is opgebouwd uit een aantal delen, namelijk:

* Het aanmaken van een test klasse waaronder de test valt
* Het aanmaken van een SetUp procudure die de tabellen compleet vrij maakt van andere constraints, zodat de constraint in isolatie te testen is
* De test gevallen, deze bestaan elk uit drie delen (Microsoft, 2016), namelijk:
  + Arrange, hierin wordt er data voorbereidt die nodig is om het testgeval uit te kunnen voeren
  + Act, hierin wordt de actie die getest moet worden uitgevoerd
  + Assert, hierin wordt gecontroleerd of de actie daadwerkelijk heeft gedaan wat er verwacht word.

Hieronder wordt een voorbeeld gegeven van de test opstelling. Voor dit voorbeeld is een deel van de test opstelling van constraint 4 genomen.

EXEC tSQLt.NewTestClass 'testSalaryGradesCantOverlap'

GO

CREATE OR ALTER PROC testSalaryGradesCantOverlap.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.grd'

EXEC tSQLt.ApplyTrigger 'dbo.grd', 'dbo.utr\_OverlappingSalaryGrades'

SELECT \*

INTO expected

FROM dbo.grd

END

GO

GO

CREATE OR ALTER PROC testSalaryGradesCantOverlap.testInsertWithWrongLowerLimit

AS

BEGIN

INSERT INTO grd VALUES (1, 10, 20, NULL)

INSERT INTO expected VALUES (1, 10, 20, NULL)

EXEC tSQLt.ExpectException @ExpectedErrorNumber = 50040

INSERT INTO grd VALUES (2, 9, 20, NULL)

EXEC tSQLt.AssertEqualsTable expected, grd

END

GO

GO

CREATE OR ALTER PROC testSalaryGradesCantOverlap.testInsertWithCorrectLowerLimit

AS

BEGIN

INSERT INTO grd VALUES (1, 10, 20, NULL)

INSERT INTO expected VALUES (1, 10, 20, NULL), (2, 11, 20, NULL)

EXEC tSQLt.ExpectNoException

INSERT INTO grd VALUES (2, 11, 20, NULL)

EXEC tSQLt.AssertEqualsTable expected, grd

END

GO

## Constraint 1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

is president earns >= 10k OK

is president earns < 10k NOT OK

is not president earns >= 10k OK

is not president earns < 10k OK

is not president || earns >= 10k

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF OBJECT\_ID('dbo.CHK\_PresidentSalary', 'C') IS NOT NULL

ALTER TABLE dbo.emp DROP CONSTRAINT CHK\_PresidentSalary

--Constraint

ALTER TABLE emp

ADD CONSTRAINT CHK\_PresidentSalary

CHECK (job <> 'PRESIDENT' OR msal >= 10000)

go

### Test

EXEC tSQLt.NewTestClass 'testEmps'

GO

CREATE OR ALTER PROC testEmps.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable @TableName = 'dbo.emp' -- Use of @SchemaName has been deprecated, use of tablename with schemaName prefixed is now prefered

EXEC tSQLt.ApplyConstraint 'dbo.emp', 'CHK\_PresidentSalary'

SELECT \*

INTO expected

FROM dbo.emp

END

GO

GO

CREATE OR ALTER PROC testEmps.testPresidentMontlySalaryShouldBeGreaterThan10000InvalidCase

AS

BEGIN

EXEC tSQLt.ExpectException @ExpectedErrorNumber = 547

INSERT INTO emp VALUES (NULL, NULL, 'PRESIDENT', NULL, NULL, NULL, 9999, NULL, NULL)

EXEC tSQLt.AssertEqualsTable expected, emp

END

GO

## Constraint 2

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

employs president employs manager employs admin OK

not employs president employs manager employs admin OK

not employs president not employs manager employs admin OK

employs president not employs manager employs admin OK

employs president not employs manager not employs admin NOT OK

employs president employs manager not employs admin NOT OK

not employs president employs manager not employs admin NOT OK

not employs president not employs manager not employs admin OK

(not employs president && not employs manager) || employs admin

-----------------------------------------------------------------------------------------

Kan misgaan als:

- Als een president/manager wordt geinsert in emp OF

- Als in emp de job van een medewerker naar president/manager wordt geupdate

en er geen administrator in de afdeling is of de geupdate medewerker de laatste administrator van de afdeling was OF

- Als in emp de laatste administrator van een afdeling wordt geupdatet naar een andere job OF

- Als in emp de departement van een president/manager wordt geupdatet naar een departement zonder administrator OF

- Als in emp de departement van de laatste administrator van een afdeling wordt geupdatet

en er een president/manager is in de oude afdeling OF

- Als in emp de laatste administrator van de afdeling gedeletet wordt en er een president/manager is in de afdeling

Er is gekozen om de update van de job te implementeren,

omdat dit evenveel gevallen zijn als updates voor het updaten van het departmentnummer (de meeste gevallen)

en er rekening gehouden moet worden dat de medewerker ook zelf de laatste administrator van een afdeling kan zijn

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GO

CREATE OR ALTER PROC usp\_UpdateEmpJob

(

@empno NUMERIC(4,0),

@job VARCHAR(9)

)

AS

BEGIN

SET NOCOUNT ON

SET XACT\_ABORT OFF

DECLARE @TranCount INT = @@TRANCOUNT

IF @TranCount > 0

SAVE TRAN ProcedureSave

ELSE

BEGIN TRAN

BEGIN TRY

DECLARE @deptno NUMERIC(2,0) = (

SELECT deptno

FROM emp

WHERE empno = @empno --PK, dus veilig om aan te nemen dat het een scalar is

)

IF (@job = 'PRESIDENT' OR @job = 'MANAGER') AND NOT EXISTS (

-- Wordt een president of manager, check of er nog een (andere, mag niet zich zelf zijn met zijn oude job) admin is

SELECT 1

FROM emp

WHERE job = 'ADMIN' AND deptno = @deptno AND empno <> @empno

)

THROW 50020, 'Er is geen admin in deze afdeling', 1

IF EXISTS (

-- Emp die geupdate wordt is een admin, dus als er een President of Manager werkt EN het de laatste admin is in de afdeling, tegenhouden

SELECT 1

FROM Emp

WHERE empno = @empno AND job = 'ADMIN'

) AND NOT EXISTS (

SELECT 1

FROM Emp

WHERE job = 'ADMIN' AND deptno = @deptno AND empno <> @empno -- Het is de laatste admin als hier true uitkomt

) AND EXISTS (

SELECT 1

FROM Emp

WHERE deptno = @deptno AND (job = 'PRESIDENT' OR job = 'MANAGER') -- En er werkt een president of manager in de afdeling

)

THROW 50021, 'Je kunt de job van de laatste admin van een afdeling waar een president of manager werkt niet veranderen', 1

UPDATE emp

SET job = @job

WHERE empno = @empno

IF @TranCount = 0 AND XACT\_STATE() = 1 COMMIT TRAN

END TRY

BEGIN CATCH

IF @TranCount = 0 AND XACT\_STATE() = 1 ROLLBACK TRAN

ELSE

BEGIN

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

END;

THROW

END CATCH

END

GO

### Test

EXEC tSQLt.NewTestClass 'testAdminInEveryDeptAPresidentOrManagerWorks'

GO

CREATE OR ALTER PROC testAdminInEveryDeptAPresidentOrManagerWorks.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp'

SELECT \*

INTO expected

FROM dbo.emp

END

GO

GO

CREATE OR ALTER PROC testAdminInEveryDeptAPresidentOrManagerWorks.testEmpToPresidentWithoutAdmin

AS

BEGIN

INSERT INTO emp VALUES (1, NULL, 'SALESREP', NULL, NULL, NULL, NULL, NULL, 10)

INSERT INTO expected VALUES (1, NULL, 'SALESREP', NULL, NULL, NULL, NULL, NULL, 10)

EXEC tSQLt.ExpectException @ExpectedErrorNumber = 50020

EXEC dbo.usp\_UpdateEmpJob @empno = 1, @job = 'PRESIDENT'

EXEC tSQLt.AssertEqualsTable expected, emp

END

GO

## Constraint 3

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. The company hires adult personnel only.

age >= 18 OK

age < 18 NOT OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF OBJECT\_ID('dbo.CHK\_employee\_age', 'C') IS NOT NULL

ALTER TABLE dbo.emp DROP CONSTRAINT CHK\_employee\_age

ALTER TABLE emp

ADD CONSTRAINT CHK\_employee\_age

CHECK (DATEADD(YEAR, 18, born) < GETDATE())

GO

Zoals u kunt zien is er ook code aanwezig om de constraint de droppen als deze al bestaat, dit is om te zorgen dat het script altijd uitgevoerd kan worden onafhankelijk van de huidige constraints in de database.

### Test

EXEC tSQLt.NewTestClass 'testEmployeesAge'

GO

CREATE OR ALTER PROC testEmployeesAge.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp'

EXEC tSQLt.ApplyConstraint 'dbo.emp', 'CHK\_employee\_age'

SELECT \*

INTO expected

FROM dbo.emp

END

GO

CREATE OR ALTER PROC testEmployeesAge.testEmpAge18OrHigherInvalidCase

AS

BEGIN

EXEC tSQLt.ExpectException @ExpectedErrorNumber = 547

INSERT INTO emp VALUES (NULL, NULL, NULL, DATEADD(YEAR, -17, GETDATE()), NULL, NULL, NULL, NULL, NULL)

EXEC tSQLt.AssertEqualsTable expected, emp

END

GO

## 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.

Kan misgaan bij:

- Insert in grd waarbij de llimit lager is dan de llimit van een lagere grd

- Insert in grd waarbij de ulimit lager is dan de ulimit van een lagere grd

- Update van grd waarbij de llimit lager wordt dan de llimit van een lagere grd

- Update van grd waarbij de ulimit lager wordt dan de ulimit van een lagere grd

Gekozen voor een trigger omdat die met dezelfde code zowel voor inserts als updates kan werken

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GO

CREATE OR ALTER TRIGGER utr\_OverlappingSalaryGrades

ON grd

AFTER UPDATE, INSERT

AS

BEGIN

BEGIN TRY

SET NOCOUNT ON

IF UPDATE(llimit) OR UPDATE(ulimit)

BEGIN

IF EXISTS (

SELECT 1

FROM inserted i

WHERE EXISTS (

SELECT 1

FROM grd g

WHERE g.grade < i.grade AND (

i.llimit < g.llimit OR

i.ulimit < g.ulimit

)

)

)

THROW 50040, 'Salary grades kunnen niet overlappen', 1

END

END TRY

BEGIN CATCH

THROW

END CATCH

END

GO

### Test

EXEC tSQLt.NewTestClass 'testSalaryGradesCantOverlap'

GO

CREATE OR ALTER PROC testSalaryGradesCantOverlap.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.grd'

EXEC tSQLt.ApplyTrigger 'dbo.grd', 'dbo.utr\_OverlappingSalaryGrades'

SELECT \*

INTO expected

FROM dbo.grd

END

GO

GO

CREATE OR ALTER PROC testSalaryGradesCantOverlap.testMultiRowInsertTestSucces

AS

BEGIN

INSERT INTO grd VALUES (1, 10, 20, NULL)

INSERT INTO expected VALUES (1, 10, 20, NULL),

(2, 15, 25, NULL),

(3, 30, 80, NULL)

EXEC tSQLt.ExpectNoException

INSERT INTO grd VALUES (2, 15, 25, NULL),

(3, 30, 80, NULL)

EXEC tSQLt.AssertEqualsTable expected, grd

END

GO

## Constraint 5

Een ander belangrijk voorbeeld is constraint 5: “*The start date and known trainer uniquely identify course offerings. Note: the use of a filtered index is not allowed.”* (HAN, z.d.) Hiervoor was namelijk al een unique constraint in de database aanwezig, na overleg met de docent, bleek dat het de bedoeling was dat dezelfde starts wel mag voorkomen wanneer trainer NULL is, maar niet als de trainer niet NULL is. Om dit te realiseren, is de oude unique constraint gedropt en is er een trigger voor in de plaats geïmplementeerd.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

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

Er mag geen nieuwe offr worden toegevoegd wanneer er al een offr bestaat met dezelfde

start tijd en trainer. Maar er mogen wel meerdere offrs worden toegevoegd met dezelfde

start datum met null als trainer.

Kan misgaan bij:

- Insert in offr waarbij de trainer en starts gelijk zijn aan een bestaande offr

- Update van offr waarbij de trainer gelijk raakt aan een bestaande offr trainer

met dezelfde start datum

- Update van offr waarbij de start datum gelijk raakt aan een bestaande offr

met dezelfde trainer

Aangezien er bij constraint 11 gebruik gemaakt wordt van een stored procedure die

er voor zorgt dat er een insert gedaan wordt in de offr tabel, is er bij deze

constraint voor gekozen om een trigger toe te passen. Dit zodat er geen

onduidelijkheid onstaat over welke stored procedure er gebruikt moet gaan

worden. Het zou ook een mogelijkheid zijn om de procedure bij 11 uit te breiden

maar voor de duidelijkheid is dit los getrokken.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

IF OBJECT\_ID('dbo.ofr\_unq', 'UQ') IS NOT NULL --Deze constraint moest verwijderd worden om vervolgens zelf een uitgebreidere versie ervan te implementeren

ALTER TABLE offr DROP CONSTRAINT ofr\_unq

GO

CREATE OR ALTER TRIGGER utr\_UniqueStartTrainer

ON offr

AFTER INSERT, UPDATE

AS

BEGIN

BEGIN TRY

SET NOCOUNT ON

IF (UPDATE(starts) OR UPDATE(trainer))

BEGIN

IF EXISTS (

SELECT 1

FROM inserted I

WHERE EXISTS (

SELECT 1

FROM offr O

WHERE I.trainer = O.trainer AND I.starts = O.starts

HAVING count(\*) > 1

)

)

THROW 50050, 'Er mogen geen offers zijn met dezelfde start datum en trainer', 1

END

END TRY

BEGIN CATCH

THROW

END CATCH

END

GO

### Test

EXEC tSQLt.NewTestClass 'testOffrIsUniqueByTrainerAndStarts'

GO

CREATE OR ALTER PROC testOffrIsUniqueByTrainerAndStarts.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr'

EXEC tSQLt.ApplyTrigger 'dbo.offr', 'utr\_UniqueStartTrainer'

SELECT \*

INTO expected

FROM dbo.offr

END

GO

GO

CREATE OR ALTER PROC testOffrIsUniqueByTrainerAndStarts.testNormalInsert

AS

BEGIN

INSERT INTO offr VALUES (NULL, '23-DEC-2019', NULL, NULL, 1, NULL)

INSERT INTO expected VALUES (NULL, '23-DEC-2019', NULL, NULL, 1, NULL),

(NULL, '24-DEC-2019', NULL, NULL, 2, NULL)

EXEC tSQLt.ExpectNoException

INSERT INTO offr VALUES (NULL, '24-DEC-2019', NULL, NULL, 2, NULL)

EXEC tSQLt.AssertEqualsTable expected, offr

END

GO

## Constraint 6

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6. Trainers cannot teach different courses simultaneously.

De starts + dur van de nieuwe offr mogen niet vallen binnen andere al gegeven offrs.

Conditie A: StartA > EndB

Conditie B: EndA < StartB

Er is overlap als geen van beide waar is

Kan misgaan als:

- Nieuwe insert wordt gedaan in offr, en de starts tijd ligt binnen een bestaande offr die gegeven wordt (starts + dur)

- Nieuwe insert wordt gedaan in offr, en de dur van de crs komt te vallen wanneer er al een andere offr wordt gegeven.

- Bij een update van de starts in offr waardoor deze binnen een andere offr komt te liggen.

- Bij een update van de duur van een course in de crs tabel waardoor deze komt te vallen binnen de duur van een ander offer

-- Deze trigger handelt geen inserts of deletes af binnen de crs tabel, daarom hier niet aan voldaan worden

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GO

CREATE OR ALTER TRIGGER utr\_OverlappingCourseOfferings

ON offr

AFTER INSERT, UPDATE

AS

BEGIN

SET NOCOUNT ON

BEGIN TRY

IF (UPDATE(starts))

BEGIN

IF exists (

SELECT \*

FROM inserted I

WHERE exists (

SELECT 1

FROM offr O

WHERE (O.trainer = I.trainer) AND (

-- StartA <= EndB

(I.starts <= DATEADD(DAY, (SELECT dur-1 FROM crs WHERE code = O.course), O.starts))

AND

-- EndA >= StartB

(DATEADD(DAY, (SELECT dur-1 FROM crs WHERE code = I.course), I.starts) >= O.starts)

) AND (O.course <> I.course AND O.starts <> I.starts)

)

)

THROW 50060, 'Een nieuwe offr mag niet binnen de tijdsduur van een al bestaande offr vallen', 1

END

END TRY

BEGIN CATCH

THROW

END CATCH

END

GO

### Test

EXEC tSQLt.NewTestClass 'testCourseOfferingsCantOverlap'

GO

CREATE OR ALTER PROC testCourseOfferingsCantOverlap.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr'

EXEC tSQLt.ApplyTrigger 'dbo.offr', 'dbo.utr\_OverlappingCourseOfferings'

SELECT \*

INTO expected

FROM dbo.offr

END

GO

GO

CREATE OR ALTER PROC testCourseOfferingsCantOverlap.testInsertWithoutErrors

AS

BEGIN

INSERT INTO offr VALUES ('PLSQL', '2006-10-08', NULL, NULL, 1016, NULL)

INSERT INTO expected VALUES ('PLSQL', '2006-10-08', NULL, NULL, 1016, NULL),

('AM4PD', '2006-10-12', NULL, NULL, 1016, NULL)

EXEC tSQLt.ExpectNoException

INSERT INTO offr VALUES ('AM4PD', '2006-10-12', NULL, NULL, 1016, NULL)

EXEC tSQLt.AssertEqualsTable expected, offr

END

GO

## Constraint 7

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

7. An active employee cannot be managed by a terminated employee.

Kan misgaan bij:

- Een update in memp waarbij de mgr naar een empno wordt gezet die in term staat

- Een insert in memp waarbij de mgr een empno is van een term

- Een insert in term waarbij de empno een mgr is in memp

- Een update in term waarbij de empno naar een mgr in memp wordt gezet

Gekozen voor een stored procedure voor het terminaten van een employee (insert in term),

omdat dit een voor de hand liggende actie is en

zodat er een default leftcomp date wordt ingevuld en een trigger geen voordelen heeft

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GO

CREATE OR ALTER PROC usp\_TerminateEmp

(

@empno NUMERIC(4),

@comments VARCHAR(60),

@leftComp DATE = GETDATE

)

AS

BEGIN

SET NOCOUNT ON

SET XACT\_ABORT OFF

DECLARE @TranCount INT = @@TRANCOUNT

IF @TranCount > 0

SAVE TRAN ProcedureSave

ELSE

BEGIN TRAN

BEGIN TRY

IF EXISTS (

SELECT 1

FROM memp m

WHERE m.mgr = @empno

AND NOT EXISTS (

SELECT 1

FROM term t

WHERE m.empno = t.empno

)

)

THROW 50070, 'Deze employee is nog een manager van een actieve employee', 1

INSERT INTO term (empno, leftcomp, comments) VALUES (@empno, @leftComp, @comments)

IF @TranCount = 0 AND XACT\_STATE() = 1 COMMIT TRAN

END TRY

BEGIN CATCH

IF @TranCount = 0 AND XACT\_STATE() = 1 ROLLBACK TRAN

ELSE

BEGIN

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

END;

THROW

END CATCH

END

GO

### Test

EXEC tSQLt.NewTestClass 'testTermEmpNotManagingActiveEmp'

GO

CREATE OR ALTER PROC testTermEmpNotManagingActiveEmp.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.term'

EXEC tSQLt.FakeTable 'dbo.memp'

EXEC tSQLt.FakeTable 'dbo.emp'

SELECT \*

INTO expected

FROM dbo.term

END

GO

GO

CREATE OR ALTER PROC testTermEmpNotManagingActiveEmp.testNormalTerm

AS

BEGIN

INSERT INTO emp VALUES (1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL),

(2, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL)

INSERT INTO memp VALUES (1, 2)

INSERT INTO expected VALUES (1, NULL, NULL)

EXEC tSQLt.ExpectNoException

EXEC dbo.usp\_TerminateEmp 1, NULL, NULL

EXEC tSQLt.AssertEqualsTable expected, term

END

GO

## Constraint 8

Het volgende voorbeeld is constraint 8: “*A trainer cannot register for a course offering taught by him- or herself.*” (HAN, z.d.) Met dit voorbeeld willen wij aantonen dat we voor elke procedure het transaction template hebben gebruikt. Dit is gedaan voor wanneer deze procedure wordt toegevoegd, er niet meer nagedacht hoeft te worden over het gebruik van transactions. De stored procedure zal zijn eigen savepoint maken wanneer er al een transaction loopt, zodat de stored procedure in zijn eigen ‘omgeving’ kan werken.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

reg.stud mag niet gelijk zijn aan offr.trainer waar de course en starts gelijk aan elkaar zijn.

Kan misgaan bij:

- Een insert in reg waarbij de stud gelijk is aan de trainer van die course

- Een update in reg zodat de stud gelijk wordt aan de trainer van die course

- Een update in offr waardoor de trainer van gelijk wordt aan een geregistreerde employee

Er is gekozen voor een stored procedure om een insert in de reg tabel te voorkomen,

omdat dit een voor de hand liggende situatie is.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GO

CREATE OR ALTER PROC usp\_InsertNewReg

(

@stud numeric(4),

@course varchar(6),

@starts date,

@eval numeric(1)

)

AS

BEGIN

SET NOCOUNT ON

SET XACT\_ABORT OFF

DECLARE @TranCount INT = @@TRANCOUNT

IF @TranCount > 0

SAVE TRAN InsertNewReg

ELSE

BEGIN TRAN

BEGIN TRY

IF EXISTS (

SELECT 1

FROM offr

WHERE trainer = @stud AND (course = @course AND starts = @starts)

)

THROW 50080, 'Er mag niet geregistreerd worden op een course offering dat gegeven wordt door dezelfde employee', 1

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

IF @TranCount = 0 AND XACT\_STATE() = 1 COMMIT TRAN

END TRY

BEGIN CATCH

IF @TranCount = 0 AND XACT\_STATE() = 1 ROLLBACK TRAN

ELSE

BEGIN

IF XACT\_STATE() <> -1 ROLLBACK TRAN InsertNewReg

END;

THROW

END CATCH

END

GO

### Test

EXEC tSQLt.NewTestClass 'testNotRegOnCourseTaughtBySameEmp'

GO

CREATE OR ALTER PROC testNotRegOnCourseTaughtBySameEmp.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp'

EXEC tSQLt.FakeTable 'dbo.reg'

EXEC tSQLt.FakeTable 'dbo.offr'

SELECT \*

INTO expected

FROM dbo.reg

END

GO

GO

CREATE OR ALTER PROC testNotRegOnCourseTaughtBySameEmp.testNormalInsert

AS

BEGIN

INSERT INTO emp VALUES (1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL),

(2, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL)

INSERT INTO offr VALUES (1016, '2006-10-07', NULL, NULL, 1, NULL)

INSERT INTO expected VALUES (2, 1016, '2006-10-07', NULL)

EXEC tSQLt.ExpectNoException

EXEC usp\_InsertNewReg 2, 1016, '2006-10-07', NULL

EXEC tSQLt.AssertEqualsTable expected, reg

END

GO

## 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.

Kan misgaan bij:

- Update in crs waarbij de duration wordt veranderd

- Update in offr waarbij de trainer wordt veranderd en dus de course ineens home based of niet meer home based is

- Update in dept waarbij de locatie wordt aangepast en dus de course ineens home based of niet meer home based is

- Update in emp waardoor de trainer in een andere locatie komt te werken en dus de course ineens home based of niet meer home based is

- Insert in offr waardoor niet meer de helft van de offerings home based is

- Delete in offr waardoor niet meer de helft van de offerings home based is

Gekozen voor trigger op offr, omdat dezelfde code dan drie van de gevallen afdekt

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE OR ALTER TRIGGER utr\_HomeBasedOfferings

ON offr

AFTER UPDATE, INSERT

AS

BEGIN

SET NOCOUNT ON

BEGIN TRY

IF UPDATE (course) OR UPDATE (trainer) OR UPDATE (loc)

BEGIN

-- Vanwege performance gekozen om te kijken welke home based zijn en dan de helft van de totale duration op te halen

-- i.p.v te vergelijken met niet homebased

IF (SELECT IIF(homeBasedDur.dur < halfDur.dur, 1, 0)

FROM (

SELECT SUM(dur) as dur

FROM offr o INNER JOIN crs c ON o.course = c.code

INNER JOIN emp e ON o.trainer = e.empno

INNER JOIN dept d ON e.deptno = d.deptno

WHERE d.loc = o.loc

) as homeBasedDur, (

SELECT SUM(dur) / 2 as dur

FROM crs c INNER JOIN offr o ON c.code = o.course

) as halfDur

) = 1

THROW 50090, 'Ten minste de helft van de offerings moet home based zijn', 1

END

END TRY

BEGIN CATCH

THROW

END CATCH

END

### Test

EXEC tSQLt.NewTestClass 'testAtLeastHalfHomeBasedOfferings'

GO

GO

CREATE OR ALTER PROC testAtLeastHalfHomeBasedOfferings.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp'

EXEC tSQLt.FakeTable 'dbo.crs'

EXEC tSQLt.FakeTable 'dbo.offr'

EXEC tSQLt.FakeTable 'dbo.dept'

EXEC tSQLt.ApplyTrigger 'dbo.offr', 'utr\_HomeBasedOfferings'

SELECT \*

INTO expected

FROM dbo.offr

INSERT INTO dbo.dept VALUES (1, NULL, 'Zutphen', NULL), (2, NULL, 'Arnhem', NULL)

INSERT INTO dbo.crs VALUES (1, NULL, NULL, 10), (2, NULL, NULL, 10), (3, NULL, NULL, 1)

INSERT INTO dbo.emp VALUES (1, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 1), (2, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 2)

END

GO

GO

CREATE OR ALTER PROC testAtLeastHalfHomeBasedOfferings.testInsertHalfHameBased

AS

BEGIN

INSERT INTO expected VALUES (1, NULL, NULL, NULL, 1, 'Arnhem'),

(2, NULL, NULL, NULL, 1, 'Zutphen')

EXEC tSQLt.ExpectNoException

INSERT INTO offr VALUES (1, NULL, NULL, NULL, 1, 'Arnhem'),

(2, NULL, NULL, NULL, 1, 'Zutphen')

EXEC tSQLt.AssertEqualsTable expected, offr

END

GO

## Constraint 10

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

Wanneer er een insert of een update wordt gedaan in de registratie tabel

Moet er gekeken worden of er courses zijn waar 6 of meer registraties op zijn.

Kan afgaan wanneer:

- Er een nieuwe insert wordt gedaan in de reg tabel

- Er een update wordt gedaan op de course in van een record in de reg tabel

Er is gekozen voor een trigger, omdat deze verandering/check gedaan kan worden na

dat er een insert/update is gedaan. Dit is niet belangrijk om te controleren

voordat de insert/update gedaan wordt.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GO

CREATE OR ALTER TRIGGER utr\_OfferingsStatusChange

ON reg

AFTER INSERT, UPDATE

AS

BEGIN

SET NOCOUNT ON

BEGIN TRY

IF (UPDATE(course))

UPDATE offr

SET status = 'CONF'

WHERE course IN (

SELECT O.course

FROM offr O JOIN reg R ON O.course = R.course AND O.starts = R.starts

WHERE O.status <> 'CONF'

GROUP BY O.course, O.starts

HAVING COUNT(\*) >= 6

) AND starts IN (

SELECT O.starts

FROM offr O JOIN reg R ON O.course = R.course AND O.starts = R.starts

WHERE O.status <> 'CONF'

GROUP BY O.course, O.starts

HAVING COUNT(\*) >= 6

)

END TRY

BEGIN CATCH

THROW

END CATCH

END

GO

### Test

EXEC tSQLt.NewTestClass 'testCourseStatusChange'

GO

CREATE OR ALTER PROC testCourseStatusChange.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.offr'

EXEC tSQLt.FakeTable 'dbo.reg'

EXEC tSQLt.ApplyTrigger 'dbo.reg', 'dbo.utr\_OfferingsStatusChange'

SELECT \*

INTO expected

FROM dbo.offr

END

GO

GO

CREATE OR ALTER PROC testCourseStatusChange.testSingleInsertToMakeNoOffrChange

AS

BEGIN

INSERT INTO offr VALUES ('PLSQL', '2006-10-08', 'SCHD', NULL, 1016, NULL)

INSERT INTO reg VALUES (NULL, 'PLSQL', '2006-10-08', NULL),

(NULL, 'PLSQL', '2006-10-08', NULL),

(NULL, 'PLSQL', '2006-10-08', NULL),

(NULL, 'PLSQL', '2006-10-08', NULL)

INSERT INTO expected VALUES ('PLSQL', '2006-10-08', 'SCHD', NULL, 1016, NULL)

EXEC tSQLt.ExpectNoException

INSERT INTO reg VALUES (NULL, 'PLSQL', '2006-10-08', NULL)

EXEC tSQLt.AssertEqualsTable expected, offr

END

GO

## 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)

Uitgegaan van job = 'TRAINER' AND (employed > 1 year OR attended course)

Kan misgaan bij:

- Update van emp waarbij de job wordt veranderd naar iets anders dan trainer

- Update van emp waarbij de hiredDate naar voren wordt gezet

- Delete van reg waarbij de gedelete record van een (nu) trainer is

- Update van reg waarbij de course of deelnemer wordt aangepast

- Insert in offr waarbij niet aan de condities wordt voldaan

Gekozen voor een sproc insert van offr, omdat dat het meest waarschijnlijke scenario is

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GO

CREATE OR ALTER PROC usp\_InsertOffering

(

@course VARCHAR(6),

@starts DATE,

@status VARCHAR(4),

@maxcap NUMERIC(2),

@trainer NUMERIC(4),

@loc VARCHAR(14)

)

AS

BEGIN

SET NOCOUNT ON

SET XACT\_ABORT OFF

DECLARE @TranCount INT = @@TRANCOUNT

IF @TranCount > 0

SAVE TRAN ProcedureSave

ELSE

BEGIN TRAN

BEGIN TRY

IF EXISTS (

SELECT \*

FROM emp

WHERE job <> 'TRAINER'

AND empno = @trainer

)

THROW 50110, 'Iemand die een course geeft moet een trainer zijn', 1

IF EXISTS (

SELECT 1

FROM emp e

WHERE e.empno = @trainer

AND (DATEDIFF(YEAR, hired, @starts) < 1)

)

BEGIN

IF NOT EXISTS (

SELECT 1

FROM emp e INNER JOIN reg r ON e.empno = r.stud

WHERE e.empno = @trainer

AND r.course = @course

)

BEGIN;

THROW 50111, 'Een trainer die nog niet 1 jaar in dienst is moet de course hebben gevolgd', 1

END

END

INSERT INTO offr VALUES (@course, @starts, @status, @maxcap, @trainer, @loc)

IF @TranCount = 0 AND XACT\_STATE() = 1 COMMIT TRAN

END TRY

BEGIN CATCH

IF @TranCount = 0 AND XACT\_STATE() = 1 ROLLBACK TRAN

ELSE

BEGIN

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

END;

THROW

END CATCH

END

GO

EXEC tSQLt.NewTestClass 'testTeacherRequirements'

GO

CREATE OR ALTER PROC testTeacherRequirements.SetUp

AS

BEGIN

EXEC tSQLt.FakeTable 'dbo.emp'

EXEC tSQLt.FakeTable 'dbo.reg'

EXEC tSQLt.FakeTable 'dbo.offr'

EXEC tSQLt.FakeTable 'dbo.crs'

SELECT \*

INTO expected

FROM dbo.offr

INSERT INTO emp VALUES (1, NULL, 'TRAINER', NULL, DATEADD(YEAR, -1, GETDATE()), NULL, NULL, NULL, NULL),

(2, NULL, 'SALESREP', NULL, DATEADD(YEAR, -1, GETDATE()), NULL, NULL, NULL, NULL),

(0, NULL, 'TRAINER', NULL, DATEADD(YEAR, -1, GETDATE()), NULL, NULL, NULL, NULL),

(4, NULL, 'TRAINER', NULL, GETDATE(), NULL, NULL, NULL, NULL),

(5, NULL, 'TRAINER', NULL, DATEADD(YEAR, -1, GETDATE()), NULL, NULL, NULL, NULL)

INSERT INTO reg VALUES (4, 'DMDD', NULL, NULL),

(3, 'DI', NULL, NULL),

(5, 'DI', NULL, NULL)

END

GO

GO

CREATE OR ALTER PROC testTeacherRequirements.testTrainerEmployedFor1YearAndHadCourse

AS

BEGIN

DECLARE @date DATE = GETDATE();

INSERT INTO expected VALUES ('DI', @date, 'CONF', 2, 5, 'Zutphen')

EXEC tSQLt.ExpectNoException

EXEC usp\_InsertOffering 'DI', @date, 'CONF', 2, 5, 'Zutphen'

EXEC tSQLt.AssertEqualsTable expected, offr

END

GO

# Opdracht C

Opdracht C luidt: *“Choose* ***two*** *of your procedural* ***constraints****. One which has no problems with non-repeatable read or phantoms, in a multi-user environment under the default isolation level (READ COMMITTED) and one which does.*

*Explain your choices by giving scenarios with two transactions that illustrate why it can or can’t go wrong. Add a success scenario with an isolation level that solves the problematic one.*

*For every scenario describe what kind of locks are acquired (e.g. s-locks and x-locks), when, why and for how long.”* (HAN, z.d.).

Voor deze opdracht hebben wij gekeken naar twee van onze constraints. Met gebruik van tabellen, waarin twee connecties / transacties zich bevinden, tonen wij aan hoe het zit met de concurrency en of dit dus fout kan gaan.

## Constraint 5: Geen concurrency problemen

Hieronder wordt er gekeken naar een constraint waarbij geen concurrency problemen op kunnen treden. Deze constraint voert een enkele update uit, en het is niet mogelijk dat er tijdens deze update een tweede connectie iets aan kan passen. Ondanks dat er dus twee keer dezelfde select statement wordt uitgevoerd, zal een andere connectie geen aanpassingen kunnen aanbrengen.

|  |  |
| --- | --- |
| Connection 1 | Connection 2 |
| SET TRANSACTION ISOLATION LEVEL  READ COMMITED |  |
| BEGIN TRANSACTION |  |
| (initialy)  insert into reg values (1111, 'RGDEV', '2004-09-03', 4) |  |
| (trigger)  UPDATE offr  SET status = 'CONF'  WHERE course IN (  SELECT O.course  FROM offr O JOIN reg R ON O.course = R.course AND O.starts = R.starts  WHERE O.status <> 'CONF'  GROUP BY O.course, O.starts  HAVING COUNT(\*) >= 6  ) AND starts IN (  SELECT O.starts  FROM offr O JOIN reg R ON O.course = R.course AND O.starts = R.starts  WHERE O.status <> 'CONF'  GROUP BY O.course, O.starts  HAVING COUNT(\*) >= 6  )  X-Lock requested and granted until end of transaction |  |
| COMMIT TRANSACTION |  |

Tabel 1 - Geen concurrency probleem

Nu blijkt dat volgens verschillende bronnen, nieuwere versies van SQL Server queries kunnen onderscheppen door het principe van: interleaving. Dit kan mogelijk tot problemen zorgen binnen de uitvoering van de select statements in deze query, maar aangezien dit buiten de course valt moet dit voor de huidige situatie geen probleem zijn. Er is meer over te lezen in een artikel van Joseph Sack (2017), of van Dmitry Piliugin (2018).

## Constraint 11: Non Repeatable Read

Hieronder wordt een situatie geschetst waarin het voor kan komen dat een employee een course gaat geven terwijl deze geen trainer is, ondanks dat er gecontroleerd wordt of dat deze employee wel een trainer is. Voor deze situatie bestaat er een employee met employee nummer 1111, en job TRAINER.

|  |  |  |
| --- | --- | --- |
| Connection 1 | Connection 2 | Employee job value |
| SET TRANSACTION ISOLATION LEVEL  READ COMMITED |  |  |
| BEGIN TRANSACTION |  |  |
| (initialy)  exec usp\_InsertOffering  'J2EE', '2050-10-10', 'CONF', 6, 1111, 'Amsterdam' |  |  |
| (stored procedure)  SELECT \* FROM emp WHERE job <> 'TRAINER'  AND empno = @trainer  S-Lock requested and granted until end of select |  | Employee 1111  Is een TRAINER |
|  | **update emp set job = 'ADMIN' where empno = 1111**  X-lock request and granted until auto commit | Employee 1111  Is een ADMIN |
| SELECT 1 FROM emp e WHERE e.empno = @trainer AND ( DATEDIFF( YEAR, hired, @starts ) < 1 )  S-Lock requested and granted until end of select |  | Employee 1111  Is een ADMIN |
| SELECT 1 FROM emp e INNER JOIN reg r ON e.empno = r.stud WHERE e.empno = @trainer AND r.course = @course  S-Lock requested and granted end of select |  | Employee 1111  Is een ADMIN |
| INSERT INTO offr VALUES ( @course, @starts, @status, @maxcap, @trainer, @loc ) |  | Employee 1111  Is een ADMIN |
| COMMIT TRANSACTION |  |  |

Tabel 2 - Non repeatable read bij constraint 11

Nadat deze transactie is uitgevoerd, is er een course offering waarbij de employee een ADMIN is, en geen TRAINER terwijl dit wel de bedoeling is volgens de constraint. Dit kan dus gebeuren wanneer er na de select controle een update gedaan wordt op dezelfde employee.

Deze situatie kan opgelost worden door de transaction isolation level te zetten naar Serializable. Wanneer dit gedaan wordt zal connectie 2 wachten tot connectie 1 zijn transactie heeft voltooid. Dit is te zien in de volgende tabel.

|  |  |  |
| --- | --- | --- |
| Connection 1 | Connection 2 | Employee job value |
| SET TRANSACTION ISOLATION LEVEL  SERIALIZABLE |  |  |
| BEGIN TRANSACTION |  |  |
| (initialy)  exec usp\_InsertOffering  'J2EE', '2050-10-10', 'CONF', 6, 1111, 'Amsterdam' |  |  |
| (stored procedure)  SELECT \* FROM emp WHERE job <> 'TRAINER'  AND empno = @trainer  S-Lock requested and granted on empno 1111 until end of transaction |  | Employee 1111  Is een TRAINER |
|  | **update emp set job = 'ADMIN' where empno = 1111**  X-lock request and not granted due to serializable isolation level in connection 1. X-lock will be granted when connection 1 has completed it’s transaction. | Employee 1111  Is een TRAINER |
| … |  |  |

Tabel 3 - Geen concurrency probleem bij constraint 11 met isolation level: serializable

Deze situatie is niet op te lossen wanneer de isolation level op repeatable read wordt gezet, aangezien de employee (1111) door de eerste select niet gevonden wordt en dus ook niet vergrendeld zal worden door de isolation level. Dit zal dus betekenen dat connectie 2 nog steeds de mogelijkheid heeft om de job van deze employee aan te passen.

# Opdracht D

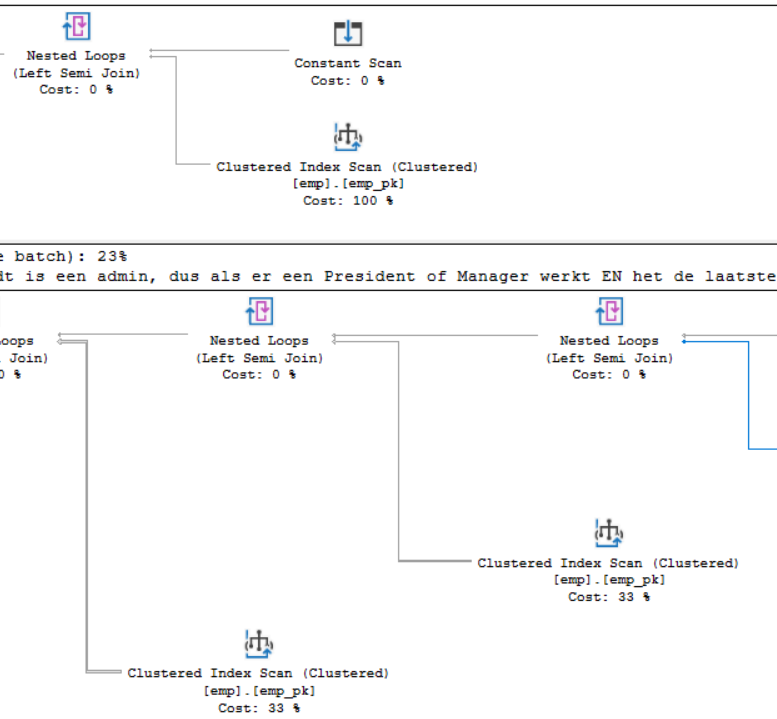
Opdracht D luidt: “*Find* ***two queries*** *to possibly optimize by adding indexes (discuss your ideas concerning the two queries to optimize with your lecturer). Use queries you wrote in task B Now think of an index (per query) on one or more columns, either clustered or non-clustered, that may optimize the query performance. Describe your solution using plain text (do not just throw a bunch of code “over the wall”) and motivate your choice of columns and clustering well. Give the code for the indexes and the execution plans before and after adding the index.*

*(Note: improvement might be hard to actually measure given this small dataset)”* (HAN, z.d.).

Om te optimaliseren is er gekeken waar er in een stored procedure of trigger geen gebruik wordt gemaakt van de primary key, waardoor een index een voordeel kan bieden.

De procedure usp\_UpdateEmpJob is een van de procedures die wij hebben gekozen, omdat deze in de emp tabel zoekt op job en op deptno, wat niet de primary key is.

Het relevante deel van het execution plan zonder extra indexes, ziet er als volgt uit:



Figuur 1 - Execution plan usp\_UpdateEmpJob zonder index

In het bovenstaande execution plan is te zien dat er drie maal een clustered index scan wordt gedaan, om dit efficienter te maken moeten dit nonclustered index seeks worden, bij voorkeur met een covering index. Hiervoor is de volgende index gebouwd:

-- Omdat er in de where zowel job als deptno worden gebruikt, zijn deze opgenomen in een nonclustered index,

-- zodat er een covering index ontstaat

-- En MS SQL ervoor kiest om een nonclustered index seek te gaan doen in plaats van een clustered index scan.

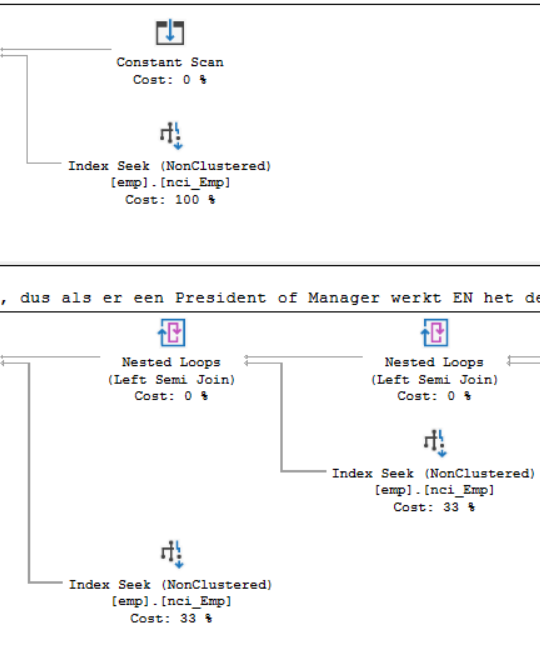
-- Op de huidige populatie is dit een vrijwel onmeetbaar verschil, echter kan het op grote populatie veel uitmaken

-- omdat deze stored procedure 2 keer gebruik maakt van deze index

CREATE NONCLUSTERED INDEX nci\_emp

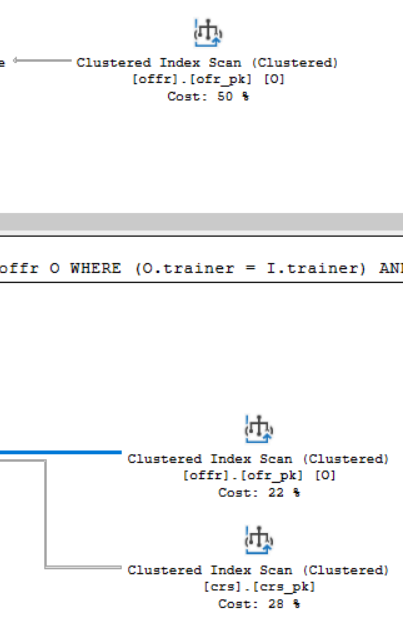
ON emp(job) include (deptno)

Het execution plan met deze index ziet er als volgt uit:



Figuur 2 - Execution plan usp\_UpdateEmpJob met covering index

De trigger utr\_UniqueStartTrainer maakt zoekt ook op kolommen die niet tot de primary key horen, namelijk starts en trainer. Het relevante deel van het execution plan ziet er als volgt uit:



Figuur 3 - Execution plan utr\_UniqueStartTrainer zonder index

Om dit efficiënter te maken, moeten de clustered index scans worden omgezet naar non clustered index seeks. De volgende index bereikt dit:

-- Door deze index wordt bij query 2 in het execution plan gekozen voor een nonclustered index seek in plaats van een clustered index scan

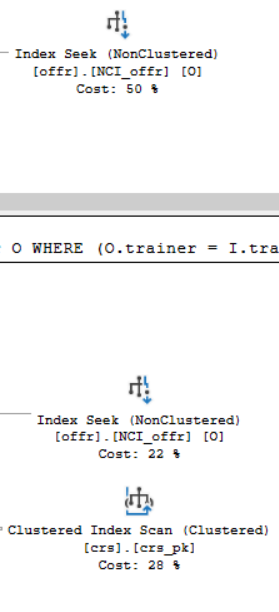
-- Bij query 3 gebeurd hetzelfde. Deze indexes zijn beide covering, waardoor de query snelheid er op vooruitgaat

-- Het is sneller om een ingang te geven op trainer dan dat het is om een ingang te geven op starts

CREATE NONCLUSTERED INDEX nci\_offr

ON offr(trainer) include(starts)

Het exection plan dat hierbij hoort ziet er als volgend uit:



Figuur 4 - Execution plan utr\_UniqueStartTrainer met index

Het is dus gelukt om middels indexes een aantal queries te optimaliseren, de complete execution plans zijn te vinden in de bijgeleverde map ‘SQL Execution Plan’.

# Opdracht E

Opdracht E luidt: “*The Course Database implements a bit of history awareness like in for instance the HIST table (changes in employee’s department and/or the salary amount are recorded by stacking the historical state in the HIST table). We want the history of all data changes to be automatically recorded in history tables belonging to the original tables using database triggers. Note: assume primary keys are immutable!*

*For every table in the Course Database a history version table needs to be available with name HIST\_<table name> so HIST\_EMP, DEPT etc.*

*These history tables versions need to have the same structure as the original tables, but with a different primary key consisting of the original primary key col-umn(s) combined with a timestamp column (type timestamp). The primary key is the only constraint of these history tables.*

*Assignment:*

*- isolate the fixed boilerplate template T-SQL trigger code and determine where you can find (using the Information Schema Views) and retrieve the parameter values you have to generate into the fixed boilerplate template T-SQL code to produce the specific trigger;*

*- write at minimum two stored procedures that do the job, one generating the trigger code per table (on the basis of an input parameter), one calling this stored procedure as many times as there are non-history tables.”* (HAN, z.d.).

Om opdracht E te implementeren is als eerste een analyse gedaan welke code er nodig is om te generen per table, hier kwam de volgende ‘predicate’ uit voort:

GO

CREATE TABLE HIST\_<tabel naam> (

ts TIMESTAMP NOT NULL,

<kolommen met datatypes>

CONSTRAINT pk\_HIST\_<tabel\_name> PRIMARY KEY (ts, <pk uit tabel>)

)

GO

CREATE TRIGGER utr\_HIST\_<tabel naam>

ON <tabel naam>

AFTER INSERT, UPDATE

AS

BEGIN

INSERT INTO HIST\_<tabel naam> (<kolommen>)

SELECT \* FROM inserted

END

GO

Om dit te realiseren zijn als eerste select statements geschreven waaruit duidelijk werd dat deze predicatie inderdaad te generen valt. Het meest uitdagende stuk code bleek het ophalen van de datatypen die bij een kolom horen te zijn, omdat hierbij rekening moet worden gehouden met in het geval van een char, nchar, varchar of nvarchar met een maximale lengte en in het geval van een numeric met de lengte en het aantal decimale plaatsen. Om aan de opdracht te voldoen is vervolgens de stored procedure usp\_generateHistTable geschreven die op basis van een table naam een geschiedenis tabel genereerd met de bijbehorende trigger. Deze procedure bestaat uit een aantal onderdelen, namelijk: het ophalen van alle kolommen van de originele tabel, het ophalen van de primary key kolommen van de originele tabel, de kolommen met bijbehorende datatypen ophalen van de originele tabel, het maken van een sql statement dat de geschiedenis tabel aanmaakt, het uitvoeren hiervan, het maken van een sql statement dat de trigger aanmaakt en het uitvoeren hiervan. Deze procedure ziet er als volgt uit:

CREATE OR ALTER PROC usp\_generateHistTable

(

@tableName VARCHAR(MAX)

)

AS

BEGIN

SET NOCOUNT ON

SET XACT\_ABORT OFF

DECLARE @TranCount INT = @@TRANCOUNT

IF @TranCount > 0

SAVE TRAN ProcedureSave

ELSE

BEGIN TRAN

BEGIN TRY

-- Alle kolomen, nodig voor de trigger

DECLARE @columns VARCHAR(MAX) = '';

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

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = @tableName

SELECT @columns = LEFT(@columns, LEN(@columns) - 1)

-- De primary kolommen, nodig om de primary key constraint aan te leggen

DECLARE @primaryKeys VARCHAR(MAX) = '';

SELECT @primaryKeys = @primaryKeys + sc.COLUMN\_NAME + ', '

FROM INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS tc

INNER JOIN INFORMATION\_SCHEMA.CONSTRAINT\_COLUMN\_USAGE sc ON tc.CONSTRAINT\_NAME = sc.CONSTRAINT\_NAME

WHERE CONSTRAINT\_TYPE = 'PRIMARY KEY'

AND tc.TABLE\_NAME = @tableName

SELECT @primaryKeys = LEFT(@primaryKeys, LEN(@primaryKeys) - 1)

-- De kolommen met datatypes

DECLARE @columnsMetDataTypes VARCHAR(MAX) = '';

SELECT @columnsMetDataTypes = @columnsMetDataTypes + c.COLUMN\_NAME +

' ' +

DATA\_TYPE +

IIF(

DATA\_TYPE LIKE '%numeric%', -- Voor numeric moet er bij komen: (NUMERIC\_PRECISION, NUMERIC\_SCALE)

'(' + CAST(NUMERIC\_PRECISION AS VARCHAR(5)) + ', ' + CAST(NUMERIC\_SCALE AS VARCHAR(5)) + ')',

IIF(DATA\_TYPE LIKE '%char%', -- Voor chars, nchar, varchar, nvarchar moet er bij komen: (CHARACTER\_MAXIMUM\_LENGTH)

'(' + CAST(CHARACTER\_MAXIMUM\_LENGTH AS VARCHAR(5)) + ')',

''

)

) +

' ' +

IIF(IS\_NULLABLE = 'NO', 'NOT NULL', 'NULL') + -- Is de kolom nullable?

', '

FROM INFORMATION\_SCHEMA.COLUMNS c

WHERE c.TABLE\_NAME = @tableName

-- Maak de tabel aan

DECLARE @sql VARCHAR(MAX) = '';

SELECT @sql = @sql + 'CREATE TABLE HIST\_' + @tableName + ' (ts TIMESTAMP NOT NULL, ' +

@columnsMetDataTypes + ' CONSTRAINT pk\_HIST\_' + @tableName + ' PRIMARY KEY (ts, ' + @primaryKeys + ')) '

EXEC(@sql)

-- Maak de trigger aan

SET @sql = '';

SELECT @sql = @sql + 'CREATE TRIGGER utr\_HIST\_' + @tableName +

' ON ' + @tableName +

' AFTER INSERT, UPDATE AS BEGIN INSERT INTO HIST\_' + @tableName +

'(' + @columns + ') SELECT \* FROM inserted END '

EXEC(@sql)

IF @TranCount = 0 AND XACT\_STATE() = 1 COMMIT TRAN

END TRY

BEGIN CATCH

IF @TranCount = 0 AND XACT\_STATE() = 1 ROLLBACK TRAN

ELSE

BEGIN

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

END;

THROW

END CATCH

END

GO

Vervolgens is de stored procedure usp\_generateAllHistory geschreven die voor alle tabellen waarvan de naam niet met HIST begint, de procedure usp\_generateHistTable uitvoert met de juiste tabelnamen. Hieronder is deze procedure opgenomen.

-- Deze functie zal voor alle tabellen de history tabel + trigger aanmaken.

GO

CREATE OR ALTER PROC usp\_generateAllHistory

AS

BEGIN

SET NOCOUNT ON

SET XACT\_ABORT OFF

DECLARE @TranCount INT = @@TRANCOUNT

IF @TranCount > 0

SAVE TRAN ProcedureSave

ELSE

BEGIN TRAN

BEGIN TRY

DECLARE @sql VARCHAR(MAX) = '';

SELECT @sql = @sql + 'EXEC usp\_generateHistTable ''' + TABLE\_NAME + ''' '

FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_NAME NOT LIKE 'HIST%'

EXEC (@sql)

IF @TranCount = 0 AND XACT\_STATE() = 1 COMMIT TRAN

END TRY

BEGIN CATCH

IF @TranCount = 0 AND XACT\_STATE() = 1 ROLLBACK TRAN

ELSE

BEGIN

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

END;

THROW

END CATCH

END

GO

# Opdracht F

Opdracht F luidt: “*Employees can register themselves for courses on offer using an app provided by the Human Resources department. The only data the employee have full access to are the data in the REG table. Of course read access to the EMP and OFFR tables are also needed (foreign key checks require the user to have access to the referenced data).   
For reporting purposes a specific service account needs to be created allowing reporting tools full read access to all data.   
- Implement a suitable (minimal and maintainable) security policy in de database, no more no less. The way you do this may depend on the way you implemented the constraints listed in task B.   
- Test your security regime. Create test users representing the user types employee and reporting service testing via “impersonation” the implementation of the security policy.”* (HAN, z.d.)

Voor het eerste deel van de opdracht hebben wij een nieuwe application role aangemaakt die volledige toegang heeft tot de REG tabel, en lees rechten voor de EMP en OFFR tabel. Hieronder is de code vermeld hoe wij dit gedaan hebben.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

F. Employees can register themselves for courses on offer using an app provided by the Human Resources department.

The only data the employee have full access to are the data in the REG table.

Of course read access to the EMP and OFFR tables are also needed

(foreign key checks require the user to have ac-cess to the referenced data).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE APPLICATION ROLE humanResourcesApp

WITH PASSWORD = 'password'

GO

USE COURSE

go

GRANT SELECT ON dbo.emp TO humanResourcesApp

GRANT SELECT ON dbo.offr TO humanResourcesApp

GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.reg TO humanResourcesApp

GO

Deze application role is getest door verschillende queries uit te voeren en te kijken of er resultaten zijn of niet. Per test wordt er als eerste voor gezorgd dat de gebruiker in de application role komt door middel van de stored procedure sp\_setapprole, het output cookie hier van opgeslagen, zodat de gebruiker na de test weer kan worden terug gezet naar de oorspronkelijke gebruiker, doormiddel van sp\_unsetapprole. In deze eerste test mag een gebruiker records selecteren uit de employee tabel, aangezien daar lees rechten op gegeven zijn.

-- Select op emp

DECLARE @cookie varbinary(8000);

EXEC sp\_setapprole 'humanResourcesApp', 'password', @fCreateCookie = true, @cookie = @cookie OUTPUT;

--Moet output geven

SELECT TOP 20 \* FROM emp

In het volgende voorbeeld mag er niet gelezen worden uit de memp tabel, aangezien hier geen rechten voor gegeven zijn.

-- Select op tabel waar dat niet mag

DECLARE @cookie varbinary(8000);

EXEC sp\_setapprole 'humanResourcesApp', 'password', @fCreateCookie = true, @cookie = @cookie OUTPUT;

--Moet error geven

SELECT TOP 20 \* FROM memp

EXEC sp\_unsetapprole @cookie

GO

Voor de tweede deel van deze opdracht hebben wij enkele user aangemaakt die volledige lees rechten heeft op alle tabellen. Dit hebben is bereikt door een reporter volledige leesrechten te geven voor het schema dbo.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For reporting purposes a specific service account needs to be created allowing reporting tools full read access to all data.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE USER reporter WITHOUT LOGIN

GO

GRANT SELECT ON SCHEMA::dbo TO reporter

GO

Ook dit tweede deel van de opdracht is getest met vergelijkbare tests zoals bij het eerste deel. Echter wordt hier gebruik gemaakt van EXECUTE AS USER, omdat het hier gaat om een user en niet om een application role. Hierbij is duidelijk gespecificeerd wat wel mag, en wat niet mag.

-- Testjes

EXECUTE AS USER = 'reporter'

SELECT TOP 20 \* FROM memp -- mag

SELECT TOP 20 \* FROM emp -- mag

SELECT TOP 20 \* FROM offr -- mag

SELECT TOP 20 \* FROM crs -- mag

SELECT TOP 20 \* FROM reg -- mag

BEGIN TRAN

-- mag allemaal niet

INSERT INTO emp VALUES (544, 'Erik', 'TRAINER', '2000-18-01', GETDATE(), 7, 7000, 'Erik', 1)

INSERT INTO offr VALUES ('DI', GETDATE(), 'CONF', 6, 1016, 'Zutphen')

UPDATE reg

SET stud = 1001

WHERE course = 'AM4DPM' AND starts = '2005-04-03' AND stud = 1000

ROLLBACK TRAN

REVERT

# Literatuurlijst

HAN (z.d.), *CASE DI*. Geraadpleegd op 27 maart 2019, van <https://onderwijsonline.han.nl/elearning/lessonfile/XyrGK5RD/eyJpdiI6IllmU0phWWRqQW16V0tTWEhWUk9DbEE9PSIsInZhbHVlIjoiRnZia09mZkpVMytRSTB6NmZNZmRraU1qMWp6SWtCY296N3o0ZVVDOXU4VjErTzlHcFpsTm1tS1Q3RFRVXC9UbXFMTDhBV3JWSDVrdHF5cmZwdFRxU3B3PT0iLCJtYWMiOiI4ODg0YTc0ZWRiMGYwZTRmNmJkNzllZjllMGY5OGE2NTVjMjk3MDMzNTc5NWZmMDZiYjhkZDRmNGViYzcyNmJlIn0=>

Microsoft (1 juli 2016), *Unit test basics*. Geraadpleegd op 27 maart 2019, van <https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2017>

Piliugin, D. (2018, 27 April). SQL Server 2017: Interleaved Execution for mTVF. Geraadpleegd van: <https://www.sqlshack.com/sql-server-2017-interleaved-execution-for-mtvf/>

Sack, J. (2017, 19 April). Introducing Interleaved Execution for Multi-Statement Table Valued Functions. Geraadpleegd van: <https://blogs.msdn.microsoft.com/sqlserverstorageengine/2017/04/19/introducing-interleaved-execution-for-multi-statement-table-valued-functions/>