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

|  |
| --- |
| Erik Knaake (598368) & Jannick Joosten (598696)  [Date] |

Inhoudsopgave

[Introductie 2](#_Toc4601852)

[Folder indeling 2](#_Toc4601853)

[Opdracht A 3](#_Toc4601854)

[Opdracht B 4](#_Toc4601855)

[Opdracht C 5](#_Toc4601856)

[Opdracht D 6](#_Toc4601857)

[Opdracht E 7](#_Toc4601858)

[Opdracht F 8](#_Toc4601859)

# Introductie

Voor het vak DI …

## Mappen structuur

In de hoofdfolder is een bestand genaamd: Di Casus.ssmssln aanwezig. 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.

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

# 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 vermeldt, 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 hoe wij de code hiervoor 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.).

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.

Een voorbeeld hiervan is:

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

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

Een deel van de bijbehorende tests zien er als volgt uit:

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

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.

# Opdracht C

# Opdracht D

# Opdracht E

# Opdracht F

# Literatuurlijst

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>

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