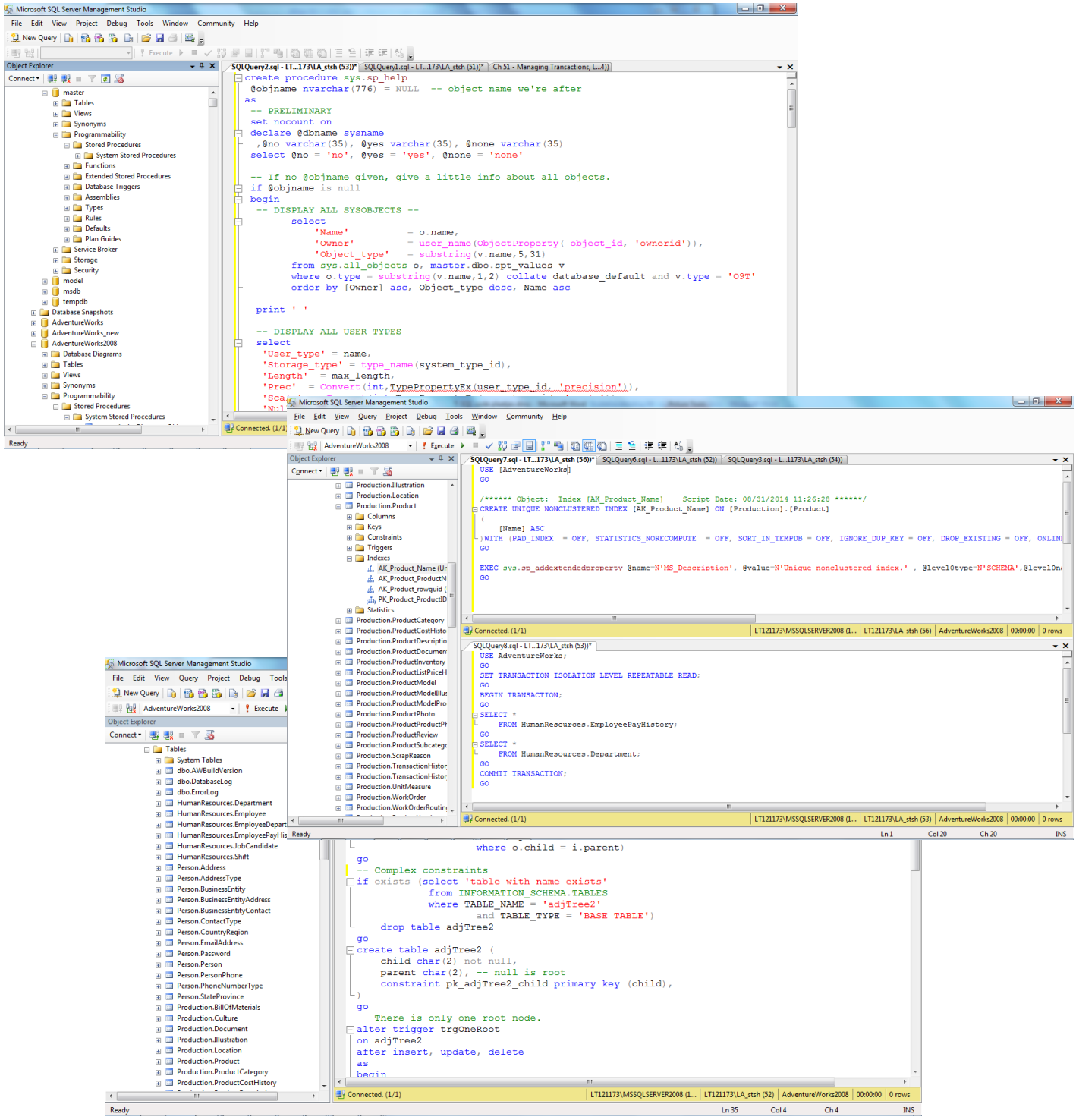
ISE-DI

Werkboek v3.1 Database Implementation 2019-2020



HAN_UNIVERSITY
OF APPLIED SCIENCES

ISE-DI

Werkboek v3.1 Database Implementation 2019-2020

Algemene gegevens

Titel	**Studentenwerkboek Database Implementation (DI)**
Studiejaar	**2018-2019**
Auteurs	**Het ISE-DI team**

HAN_UNIVERSITY
OF APPLIED SCIENCES

ISE-DI

Werkboek v3.1 Database Implementation 2019-2020

INHOUDSOPGAVE

1 Thema Herhaling SQL

Leerdoelen

Zie de competenties en beoordelingscriteria over SQL van de propedeusecourse Databases & Applications.

Workshop

- Oefen met de **SQL Server Management Studio** (SSMS), Books Online, en SQL. SSMS is voor developers of database administrators (DBA's) de belangrijkste client voor SQL Server. Raadpleeg zonodig [JORGENSEN] Chapter 5.
- Installeer de **Muziekdatabase** (uitgebreide versie), zie scripts op OnderwijsOnline.

Zelfstudie

- **Herhaling SQL.**
  Bestudeer uit [WIEGE] de in de propedeuse behandelde stof.

  Bestudeer ook in [JORGENSEN] de onderdelen die overeenkomen met de in de propedeuse behandelde stof. Die stof is verspreid over verschillende hoofdstukken. Hieronder volgt een leeswijzer.
  Chapter 6
  WHERE: pag. 120 t/m 128. Sla de verhalen over indexen over, die komen later aan de orde.
  Columns, Stars, Aliases, and Expressions: pag. 129 t/m 131.
  Ordering the Result Set: pag. 131 t/m135 (sla collation over).
  SELECT DISTINCT: pag. 137 t/m 139.
- ***Chapter 8***
- Data Types, Expressions: pag. 173 t/m 183.
- Working with NULLs: pag. 185 t/m 186.
- ***Chapter 9***
- JOINs: pag. 213-233.
- Simple subqueries: pag. 237 t/m 238, 241 t/m 243.
- ***Chapter 10***
- Standard aggregates, GROUP BY : pag. 249 t/m 253.
- HAVING: pag. 256.
- ***Chapter 11***
- Views: Chapter 271 t/m 278.
- ***Chapter 12***
- INSERT/VALUES, INSERT/SELECT: pag. 291 t/m 297.

- SELECT INTO: pag. 300 t/m 302.
- UPDATE: pag. 302 t/m 305.
- DELETE: pag. 310 t/m 312.
- **Opdrachten Herhaling SQL**
  Maak de opdracht(en) zoals hieronder aangegeven.

1.1 Herhalingsopdrachten SQL

De volgende opdrachten hebben betrekking op de Muziekdatabase. Scripts om de database aan te maken kun je op Onderwijs Online vinden.

Geef een SQL SELECT statement voor elk van de onderstaande informatiebehoeften.

1. Geef voor elk klassiek stuk het stuknr, de titel en de naam van de componist.
2. Welke stukken zijn gecomponeerd door een muziekschooldocent? Geef van de betreffende stukken het stuknr, de titel, de naam van de componist en de naam van de muziekschool.
3. Bij welke stukken (geef stuknr en titel) bestaat de bezetting uit ondermeer een saxofoon?
   Opmerking: Gebruik een subquery.
4. Bij welke stukken wordt de saxofoon niet gebruikt?
5. Bij welke jazzstukken worden twee of meer verschillende instrumenten gebruikt?
6. Geef het aantal originele muziekstukken per componist. Ook componisten met nul originele stukken dienen te worden getoond.
7. Geef voor elk niveau de niveaucode, de omschrijving en het aantal klassieke speelstukken.
   Opmerking: Dus ook als er voor een niveau geen klassieke speelstukken zijn.
8. Geef het nummer en de naam van de muziekscholen waarvoor meer dan drie speelstukken bestaan die gecomponeerd zijn door docenten van de betreffende school.
9. Voorspel uit hoeveel rijen het resultaat van de volgende query bestaat:

```
SELECT *
FROM   Componist, Muziekschool;
```

Ga vervolgens na of je voorspelling correct is.

Opmerking: Deze query heeft hetzelfde effect als

```
SELECT *
FROM   Componist CROSS JOIN Muziekschool;
```

10. Stel de tabel Componist is gedefinieerd zonder een UNIQUE-constraint op de kolom naam. Als je deze constraint toevoegt m.b.v. een ALTER TABLE statement, dan lukt dat niet als er dubbele waarden voorkomen in de kolom naam.
    Geef een SELECT-statement waarmee je de rijen met een niet-unieke componistnaam kunt opsporen.
11. Geef twee UPDATE-statements waarmee alle stukken van docenten van Muziekschool Sonsbeek op niveaucode C gezet worden, als de niveaucode not null was: gebruik Ms SQL Server's multiple tables UPDATE optie beschreven in the SQL Bible op pagina's 302 tm 305 en scrhijf een ANSI SQL

subquery variant.

Plaats je statements tussen een BEGIN TRANSACTION - ROLLBACK TRANSACTION statement combinatie. Daarmee worden wijzigingen aan de data uiteindelijk weer ongedaan gemaakt terwijl resultaten tijdelijk wel beschikbaar zijn voor controle op correctheid.

```
BEGIN TRANSACTION
   <jouw UPDATE/DELETE statement>
   <jouw SQL SELECT controle statement>
ROLLBACK TRANSACTION
```

12. Geef twee DELETE-statements waarmee alle bezettingsregels van stukken van docenten van Muziekschool Sonsbeek verwijderd worden: gebruik Ms SQL Server's multiple tables DELETE optie beschreven in the SQL Bible op pagina's 310 tm 312 en scrhijf een ANSI SQL subquery variant.

    Gebruik weer onderstaande transactie statements.

    ```
    BEGIN TRANSACTION
       <jouw UPDATE/DELETE statement>
       <jouw SQL SELECT controle statement>
    ROLLBACK TRANSACTION
    ```

    Meer over transacties volgt in in het thema Concurrency.

2 Thema Advanced SQL

Leerdoelen

Zie competenties DI-1, DI-2 in de studiehandleiding.

Theorieles en workshop

Onderwerpen Advanced SQL:

- Gecorreleerde subquery's
- De EXISTS operator
- De UNION operator
- De EXCEPT operator
- De INTERSECT operator
- Logical query processing
- De ON clause

Workshop

- Werken aan de opgaven met de **EXISTS** operator, zie onder.
- Installeer **AdventureWorks** (**OLTP**, OnLine Transaction Processing), één van de sample databases van Microsoft. Deze sample databases worden in bijna alle publicaties over MS SQL Server gebruikt.
  AdventureWorks is te vinden via www.codeplex.com (url 22 augustus 2015). Zoekterm: *sql server sample databases*. Een backup van deze database kun je ook vinden op OnderwijsOnline in folder Algemene Informatie/Databases.

Zelfstudie

- **Advanced SQL.**
  - Bestudeer uit [JORGENSEN] pag. 233 t/m 236 (voor de UNION , INTERSECT en EXCEPT operators), en pag. 244 t/m 246 voor correlated subqueries.
  - Bestudeer uit [WIEGE] de EXISTS operator en correlated subqueries, en pag. 182 t/m 187 voor de set operators.
  - Zie ook de tijdens de lessen gebruikte sheets.
- **Opdrachten Advanced SQL (2.1)**
  Maak de opdracht(en) over de EXISTS operator en gecorreleerde subquery's, zie onder.
- **Opdrachten SET operators (2.5)**
  Maak de onderstaande opdrachten om met de SET operators te oefenen.
- **Andere opdrachten (2.2 t.m. 2.4)**
  Zoek/verzin zelf voorbeelden na bestuderen van slides en Bible.
- **Opdrachten SQL voor smarties (2.6 = ni**et verplichte oefenstof).
  Als je deze opdrachten maakt krijg je SQL echt onder de knie. Je kunt ze oplossen zoals je wilt.

2.1 Opdrachten EXISTS operator en gecorreleerde subquery's

Gebruik in de volgende opdrachten de EXISTS-operator.

1. Geef stuknr en titel van elk stuk waar een piano in meespeelt.

2. Geef stuknr en titel voor elk stuk waar géén piano in meespeelt.
3. Geef instrumenten (instrumentnaam + toonhoogte) die niet worden gebruikt.
4. Geef componistId en naam van iedere componist die meer dan 1 stuk heeft gecomponeerd.
5. Geef alle originele stukken waar geen bewerkingen van zijn.
6. Geef de drie oudste stukken (zonder top te gebruiken).
7. Is er een stuk waarin alle instrumenten meespelen?
8. (Geen EXISTS) Maak een lijst van stukken, gesorteerd op lengte, en zorg voor een rangnummer (waarbij stukken van gelijke lengte hetzelfde rangnummer krijgen).
9. Geef een query voor de volgende informatiebehoefte:
   Geef geordende paren van stukken die precies dezelfde bezetting hebben. Ook als beide stukken geen bezettingsregels hebben.
   Voorbeeldoutput van de gevraagde query bij de oorspronkelijke populatie van de Muziekdatabase:

```
stuknr                 stuknr
-------------------------------------- ----------------------
1                      8
1                      10
8                      10
```

Er zijn namelijk in de oorspronkelijk gegeven populatie geen stukken met bestaande bezettingsregels die precies hetzelfde zijn. Daarom moeten er bij de oorspronkelijke populatie die stukken uitkomen die helemaal geen bezettingsregels hebben: stukken 1, 8 en 10.

Maar stel dat we een stuk toevoegen met stuknr 16 , en met dezelfde bezettingsregels als stuknr 12, door het uitvoeren van de volgende inserts:

```
INSERT INTO Stuk VALUES (16, 10, 'Blue blue', 1,  'jazz',   'A', 4,  1998)
INSERT INTO Bezettingsregel VALUES (16, 'piano',  '',   1)
INSERT INTO Bezettingsregel VALUES (16, 'fluit',  '',   2)
```

Dan moet de gevraagde query als resultset geven:

```
stuknr                 stuknr
-------------------------------------- ----------------------
1                      8
1                      10
8                      10
12                     16
```

2.2 Opdracht CASE statement

Bestudeer dit statement in [JORGENSEN] op pag. 181 t/m 183 en in Books Online, en maak zelf een voorbeeld.

2.3 Opdracht system function ISNULL()

Bestudeer deze functie in [JORGENSEN] op pag. 188 en in Books Online, en maak zelf een voorbeeld.

2.4 Opdracht data-type conversion system functions CAST() en CONVERT()

Bestudeer deze functies in [JORGENSEN] op pag. 207 t/m 210 en in Books Online, en maak zelf een voorbeeld.

2.5 Opdrachten SET operators

Voer het volgende script uit.

```
USE MASTER
GO
DROP DATABASE BEIJNG2015
GO
CREATE DATABASE BEIJNG2015
GO
USE BEIJNG2015
GO
CREATE TABLE ALL_TIME_OUTDOOR_TOP_LIST(
       RESULT VARCHAR(5),
       NAME VARCHAR(50) PRIMARY KEY)
GO
INSERT INTO ALL_TIME_OUTDOOR_TOP_LIST
VALUES ('21.34','Florence GRIFFITH-JOYNER (USA)'    ),
              ('21.62','Marion JONES (USA)'                    ),
              ('21.63','Dafne SCHIPPERS (NED)'            ),
              ('21.64','Merlene OTTEY (JAM)'                   ),
              ('21.66','Elaine THOMPSON (JAM)'            ),
              ('21.69','Allyson FELIX (USA)'                   ),
              ('21.71','Marita KOCH (GDR)'                     ),
              ('21.71','Heike DRECHSLER (GDR)'            ),
              ('21.72','Grace JACKSON (JAM)'                   ),
              ('21.72','Gwen TORRENCE (USA)'                   )

CREATE TABLE SEASON_OUTDOOR_TOP_LIST(
       RESULT VARCHAR(5),
       NAME VARCHAR(50)  PRIMARY KEY)
GO
INSERT INTO SEASON_OUTDOOR_TOP_LIST
VALUES ('21.63','Dafne SCHIPPERS (NED)'             ),
              ('21.66','Elaine THOMPSON (JAM)'            ),
              ('21.97','Veronica CAMPBELL-BROWN (JAM)'    ),
              ('21.98','Allyson FELIX (USA)'                   ),
              ('22.01','Candyce MCGRONE (USA)'            ),
              ('22.07','Dina ASHER-SMITH (GBR)'                ),
              ('22.14','Shaunae MILLER (BAH)'                  ),
              ('22.18','Dezerea BRYANT (USA)'                  ),
              ('22.20','Jenna PRANDINI (USA)'                  ),
              ('22.23','Tori BOWIE (USA)'                              ),
              ('22.23','Jeneba TARMOH (USA)'                   )
```

A. Geef een lijst met alle bekende hardloopsters.
B. Welke hardloopsters staan in de ALL-TIME highscore maar hebben die tijd niet dit seizoen gelopen?
C. Welke hardloopsters hebben dit seizoen een tijd gelopen waardoor ze in de ALL-TIME highscore lijst zijn gekomen? (veronderstel dat ze deze tijd maar 1 keer hebben gelopen).
D. Welke hardloopsters staan maar in 1 van beide lijsten?

2.6 Opdrachten SQL voor smarties (*geen* tentamensof; gebruik van SQL windowing en ranking functies)

A1. Geef van het oudste stuk/stukken het stuk/stukken met de meeste bezettingsregels? Geef stuknr, titel, jaartal en aantal bezettingsregels.

A2. Geef alle stukken die evenveel bezettingsregels hebben als deze oudste stuk/stukken met de meeste bezettingsregels.

B1. Welk stuk is het middelste stuk na ordening op stuknr. (Je mag er vanuit gaan dat er een oneven aantal stukken is).

B2. Welk stukken zijn de middelste twee stukken na ordening op stuknr. (Je mag er vanuit gaan dat er een even aantal stukken is).

B3. Geeft het middelste indien er een oneven aantal stukken is, indien er een even aantal stukken is, geef dan de middelste twee.

C. Geef de scholen (naam en plaats) met de meest gecomponeerde muziekstukken.

D. Geef per letter (A..Z) hoeveel stukken met deze letter als titel beginnen.

E. Geef de componisten gesorteerd per muziekschool en het totaal aantal componisten per muziekschool als extra rij (zogenaamde group footer per muziekschool).
Ook de duplicaten (school) moeten verborgen worden.
Dus zoiets:

```
School Componist
x      12
x      31
x      46
x      totaal: 3
y      11
y      34
y      totaal: 2
z
z      totaal: 0
```

F. Geef alle componisten, toon naam, geboortedatum, geboortedag en geboorte-seizoen.
Sorteer allereerst op geboorte-eeuw en daarbinnen
voor de geborenen eerder dan de huidige eeuw op geboortedag, in de volgorde van maandag tm zondag
voor de geborenen in deze eeuw op seizoen, in de volgorde winter, lente, zomer, herfst

Neem ook de geboortedag en het seizoen op in de output ter controle.

Dus zoiets:

```
Componist    Geboortedatum      Geboortedag  Seizoen
45           7-1-1812           maandag
11           1-1-1812           zondag
12           1-1-2012                        winter
10           1-1-2013                        winter
60           1-8-2012                        zomer
```

3 Thema Constraints

Leerdoelen

Zie competentie DI-2 in de studiehandleiding.

Theorieles en workshop

Onderwerpen:

- Welke soorten constraints zijn er.
- Wat is **declaratieve** en **procedurele constraintbewaking**.

Workshop:

Werken aan de opgaven en aan de casus.

Zelfstudie

- **Constraints.**
  - Bestudeer de sheets.
  - Bestudeer uit [WIEGE] de in de propedeuse behandelde stof. (PK'S, FK.S, checks, null/not null, ak's, defaults)
  - De stof bestaat voornamelijk uit herhaling van onderwerpen uit de propedeuse.
- De hoofdstukken uit [WIEGE] behandelen hoe je gegevensintegriteit op een *declaratieve* manier kunt bewaken, en is dus een van de belangrijkste hoofdstukken. Wat constraints waren wist je natuurlijk al, maar nu wordt er in een breder verband naar gekeken. *Als* je een constraint op een **declaratieve** manier kunt bewaken, moet je dat **altijd** doen: door triggers of stored procedures bewaakte constraints worden altijd langzamer uitgevoerd en zijn ook lastig of onmogelijk te genereren met een case tool. Daarentegen hebben case tools i.h.a. geen enkel probleem om pk's, fk's, ak's, check's, default's en dergelijke te genereren.

  Bedenk dat constraints een *naam* hebben, dus ook PK en FK constraints. Maak een tabel met een foreign key en een check constraint die je geen namen geeft en bekijk in de Object Explorer van SSMS de gegenereerde namen. Op grond van wat je dan ziet zou je moeten concluderen dat je maar beter zelf namen kunt geven aan constraints.

  Oefen ook met het ALTER TABLE statement dat je nogal eens nodig hebt.

  Besteed extra aandacht aan de pagina's 312, 313 over *cascading updates and cascading deletes*. In SQL Server 2012 is het mogelijk cascading actions declaratief te specificeren, maar niet in alle gevallen. Bijvoorbeeld wanneer een foreign key verwijst naar de tabel waar hij zelf in voorkomt, zoals bij een tabel *Employee* waarin een kolom verwijst naar de mogelijke manager van de employee. In die gevallen moeten cascading actions dus procedureel worden bewaakt met een trigger of stored procedure.

  Wees je zeer bewust van het feit dat foreign keys een *bi-directional* karakter hebben. Bij het declaratief specificeren van een fk zijn er vier mogelijke referential integrity actions mogelijk: NO ACTION, CASCADE, SET NULL, en DEFAULT. Als je niets opgeeft is NO ACTION de default.

Vaak wordt over het hoofd gezien dat tabellen *alternate keys* (alternatieve of kandidaatsleutels) kunnen hebben die met een UNIQUE kunnen worden bewaakt.

CHECK constraints zijn zeer belangrijk, en kunnen in veel gevallen worden toegepast, ook in combinatie met system of zelfs met user defined functions, zoals we later zullen zien.

3.1 Opdracht declaratieve constraints

1. Maak de tabel *Product* aan volgens de volgende specificatie:

Product			
ProductCode	int	<pk>	not null
ProductName	varchar(40)		not null
ProductPrice	money		not null
ProductsInStock	smallint		not null

Voeg na het aanmaken van de tabel een CHECK-constraint toe die de volgende business rule bewaakt:
Als de waarde van ProductName gelijk is aan TELEVISION dan moet de waarde van ProductPrice minstens 200 zijn.

2. Maak de tabel *Person* aan volgens de volgende specificatie:

```
IF EXISTS(SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME = 'Person' AND
TABLE_TYPE = 'BASE TABLE')

DROP TABLE Person
GO
CREATE TABLE Person
(ID         INT IDENTITY   NOT NULL,
 Name              VARCHAR(30)     NOT NULL,
 Prefix            VARCHAR(10)     NULL,
 NamePartner       VARCHAR(30)     NULL,
 PrefixPartner     VARCHAR(10)     NULL,
 Presentation      INT             NOT NULL,-- 1 = Name, 2 = Name + Partner, 3 = Partner
+ Name, 4 = Partner
Sex         CHAR(1) NOT NULL,      -- M = Male, F = Female
 Initials          VARCHAR(10) NOT NULL,
 CONSTRAINT PK_Person PRIMARY KEY (ID)
)

GO
```

Voeg na het aanmaken van de tabel een CHECK-constraint toe die de volgende business rule bewaakt:
Als de waarde van Presentation ongelijk is aan 1 dan moet de waarde van NamePartner bekend zijn.
Voeg dan de volgende records toe:

```
INSERT INTO dbo.Person
(Name,Prefix,NamePartner,PrefixPartner,Presentation,Sex,Initials) VALUES
    ('LastName1',NULL,NULL,NULL,1,'M','AB')
INSERT INTO dbo.Person
(Name,Prefix,NamePartner,PrefixPartner,Presentation,Sex,Initials) VALUES
    ('LastName2',NULL,'Partner2',NULL,2,'M','CD')
INSERT INTO dbo.Person
(Name,Prefix,NamePartner,PrefixPartner,Presentation,Sex,Initials) VALUES
    ('LastName3','van der','Partner3',NULL,3,'F','EF')
```

```
INSERT INTO dbo.Person
(Name,Prefix,NamePartner,PrefixPartner,Presentation,Sex,Initials) VALUES
   ('LastName4','van der','Partner4','in ''t',4,'F','GH')
```

Test de check constraint met 4 zinvolle inserts. Laat deze beoordelen door je docent.

3. De navolgende business rules dien je te implementeren met behulp van constraints op de database van de centrale casus van de course Database Implementation. De noodzakelijke scripts kun je vinden op OnderwijsOnline in folder Algemene Informatie/Casus Database Implementation.

   Een salarisschaal specificeert een salarisinterval van minimaal 500 euro.

   De ondergrenswaarde van een salarisschaal (llimit) identificeert de schaal in kwestie.

   Een cursus uitvoering (tabel OFFR) heeft altijd een trainer tenzij de status waarde aangeeft dat de cursus afgeblazen (status 'CANC') is of dat de cursus gepland is (status 'SCHD').

   Test elk van de constraints met zinvolle inserts (na week 3 doen we dit anders).

4 Thema Transact-SQL

Leerdoelen

Zie competenties en beoordelingscriteria DI-3, DI-9 in de studiehandleiding.

Theorieles en workshop

Onderwerpen:

- T-SQL Data types
- Selection, iteration

Workshop:
Werken aan de opgaven en aan de casus.

Zelfstudie

- Bestudeer uit [JORGENSEN]:
  - Data types: Chapter 8 pag 173-176. En in de Object Explorer van Management Studio de *SQL Server Data Types* (zoek bij een database onder *Programmability*).
  - Chapter 16 pag. 389-403.
- Bestudeer de sheets.

T-SQL is een uitbreiding (extension) van SQL.

Het begrip *batch* wordt in dit hoofdstuk behandeld. Dit is een belangrijk begrip, dat vaak verward wordt met het begrip *transactie*, waarover later meer.

T-SQL kent variabelen, selectie- en iteratie- statements. Alle variabelen zijn *local*, en de naam van een variabele moet beginnen met @. Maar beter niet met @@: SQL Server laat sommige zogenaamde *system functions zonder parameters* beginnen met @@. Enkele daarvan moet je nu al kennen: @@ROWCOUNT, @@ERROR.

Pas op met het gebruik van @@ERROR. Bedenk dat @@ERROR *altijd* de waarde geeft die door het laatste statement werd veroorzaakt. Maak dus altijd een eigen lokale variabele aan waarin je de waarde van @@ERROR bewaart.

Kijk ook vast eens naar de lijst van functies in Books Online, of in de Object Explorer van Management Studio bij een database onder de folder Programmability: SQL Server biedt zeer veel functions aan die de moeite waard zijn om te gebruiken.

4.1 Opdrachten Transact SQL

1. Gegeven is een bankrekeningnummer van 9 cijfers, bijvoorbeeld 972428577. Maak een T-SQL script waarmee je kunt controleren of dat een geldig bankrekeningnummer is volgens de modulo11-check. 972428577 is geldig, want:
   ```
   (9*9 + 8*7 + 7*2 + 6*4 + 5*2 + 4*8 + 3* 5 + 2*7 + 1*7) % 11 = 0
   ```
2. Bestudeer in [JORGENSEN] het CASE-statement, vanaf pagina 181. Maak een script waarmee je de naam van de huidige dag in het nederlands krijgt, gebruik makend van CASE.

5 Thema Stored procedures

Leerdoelen

Zie Competenties en beoordelingscriteria DI-3, DI-9 in de studiehandleiding.

Theorieles en workshop

Onderwerpen:

- Error handling met TRY/CATCH, RAISERROR(), THROW
- Parameters van sp's
- Return values van sp's
- Performance

Workshop: oefeningen.

Zelfstudie

Stored procedures.

- [JORGENSEN] Chapter 16 pag. 412-415, 418-424.
- [JORGENSEN] Chapter 17 pag. 431-439, pag. 444 - 448 tot aan *WITH RESULT SETS*, pag. 450 - 451.
- Zie ook de tijdens de lessen gebruikte sheets.

Wen je aan om bij aanroep van een sp altijd de namen van de parameters mee te geven. Dit is belangrijk voor het onderhoud.

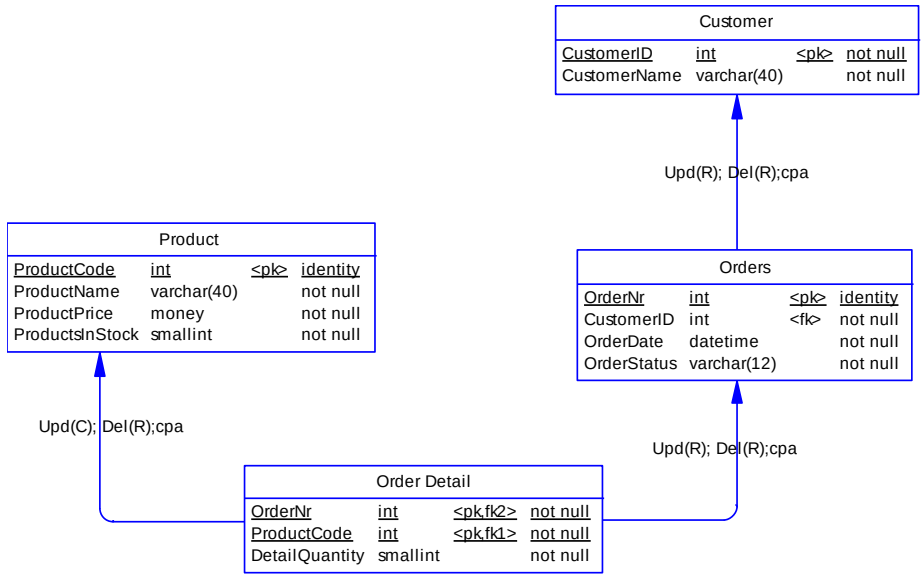
Bij het gebruik van OUTPUT parameters wordt vaak de fout gemaakt om het keyword OUTPUT weg te laten bij de aanroep van de sp. Bovendien wordt vaak vergeten de parameter expliciet uit te lezen door middel van een SELECT.

Return values zijn bedoeld om de caller van de sp iets te melden over het slagen of falen van de sp. Helaas worden ze nogal eens misbruikt om integer data terug te geven; gebruik hier OUTPUT parameters voor.

5.1 Opdracht stored procedures

Alle opgaven moeten worden gemaakt via de SQL Server Management Studio, dus niet m.b.v. PowerDesigner of een ander tool.

1. Bestudeer het onderstaande LRS:



2. Maak in je database tabellen aan die dit LRS correct implementeren en vul deze met een voorbeeld populatie, wat voor deze opgave ook mag met DRI (declarative referential integrity). Test de implementatie.

3. *Stored Procedure zonder parameters*. Schrijf een stored procedure die voor alle records uit Orders die als OrderStatus Ready hebben de CustomerID, de CustomerName, het OrderNr en de OrderStatus geeft.
   Test de procedure.

4. *Stored Procedure met parameters*. Schrijf een stored procedure die, gegeven een OrderStatus, alle records geeft uit Orders met die OrderStatus. Geef de CustomerID, de CustomerName, het OrderNr en de OrderStatus.
   Test de procedure.

5. *Stored Procedure met parameters en output parameters*. Schrijf een stored procedure die gegeven een CustomerName en een OrderStatus het aantal orders teruggeeft dat die customer heeft met die OrderStatus.
   Test de procedure.

6. Maak een stored procedure die bij een insert checkt of een kolom wel een not null waarde heeft.
   Is dit een zinnige sp? Verklaar je antwoord.

7. Maak een stored procedure met één parameter @sortParameter van type varchar die:
   a. Als @sortParameter = ‘OrderDate’ alle records uit Orders geeft gesorteerd op column ‘OrderDate’
   b. Als @sortParameter = ‘OrderStatus’ alle records uit Orders geeft gesorteerd op column ‘OrderStatus’
   c. In alle andere gevallen een foutmelding geeft.

8. De navolgende requirements dien je te implementeren met behulp van stored procedures in de database van de centrale casus van de course Database Implementation.

Houdt er rekening mee dat meer acties op de data business rules kunnen schenden (zoals combinaties van verwijderen, toevoegen en veranderen van de data).

Jullie kennis op het gebied van database programmeren is nu nog beperkt. Implementeer in deze week 2 daarom constraint logica en basic error handling. Andere toeters en bellen komen er later nog/wel bij.

a) Een manager kan niet meer dan 2 afdelingen managen.
b) Er is maar maximaal één werknemer de president van het bedrijf in kwestie.
c) Een verkoop medewerker kan niet meer verdienen dan zijn baas (houdt rekening met de commissie die ook tot het salaris wordt gerekend en een jaarlijkse commissiebedrag betreft, terwijl het reguliere basissalaris een maandsalaris betreft).
d) Een trainer kan geen cursus geven voor zijn/haar datum in dienst treding.
e) Een manager moet tevens werknemer zijn van de afdeling die zij/hij bestuurt.

9. De navolgende requirement dien je te implementeren met behulp van stored procedures in de database van de centrale casus van de course Database Implementation.

   Wijzigingen van salarissen (commissie en maandsalaris) moeten gelogd worden in een logtabel. In die tabel registreren we wie (de user), wanneer, welke wijziging heeft aangebracht. Ontwerp een tabel en schrijf een update stored procedure die deze functionaliteit implementeert.

CASUS

Werk verder aan de Casus.

6 Thema Triggers

Leerdoelen

Zie Competenties en beoordelingscriteria DI-4, DI-5, DI-9 in de studiehandleiding.

Theorieles en workshop

Onderwerpen:

- After triggers
- Rollback in een trigger
- Toepassingen van triggers.

Workshop: oefeningen

Zelfstudie

- **After triggers.**
  - Bestudeer uit [JORGENSEN] Chapter 36. Overslaan paragrafen met titels *Instead of Triggers*, *Listing Triggers, Multiple-Trigger Interaction*, *DDL Triggers*
  - Zie ook de tijdens de lessen gebruikte sheets.

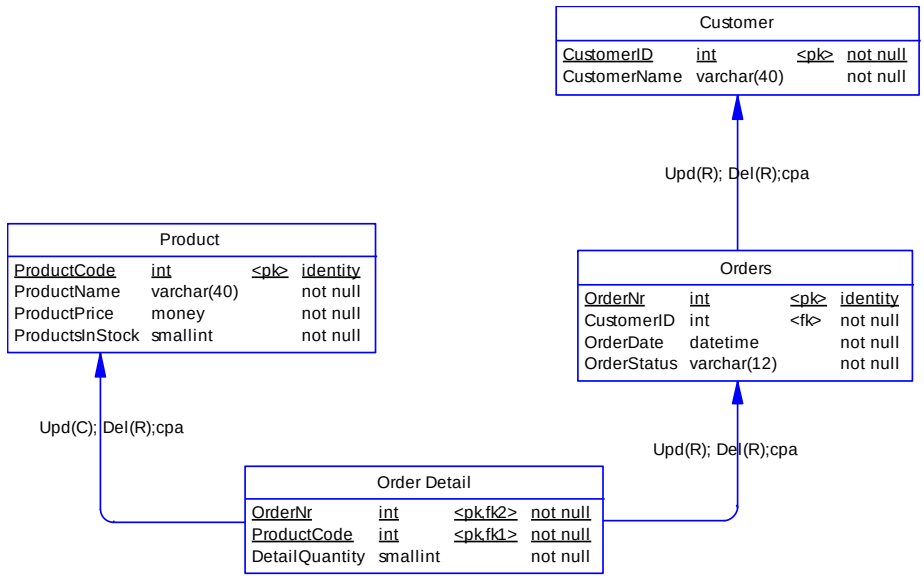
In SQL Server 2012 zijn er naast AFTER triggers ook INSTEAD OF triggers te maken. In deze course maken we van die laatste géén gebruik.

In voorgaande versies van SQL Server werden vaak triggers gebruikt om cascading updates en deletes mogelijk te maken. In SQL Server 2012/2014 is het in de *meeste* gevallen mogelijk dit te regelen via DRI (declaratieve referentiële integriteit), maar niet altijd! Het is dus niet onwaarschijnlijk dat je dit mechanisme toch nodig zult hebben. In een opdracht kun je dit oefenen.

Het is mogelijk meerdere triggers te definiëren voor één event. Er zijn dus bijvoorbeeld meerdere update triggers mogelijk op een tabel, die allemaal ‘afgaan’ als de update slaagt. Er is helaas weinig controle mogelijk op de ‘firing order’ van de triggers. Ons advies is dan ook geen gebruik te maken van deze mogelijkheid. Maar als je een trigger vergeet te droppen en je maakt een andere aan met een andere naam, dan zijn er opeens twee triggers waardoor je voor onaangename verrassingen kunt komen te staan. Dus je moet je wel bewust zijn van deze mogelijkheid.

6.1 Opdracht after triggers

Alle opgaven moeten worden gemaakt via de SQL Server Management Studio, dus niet m.b.v. PowerDesigner of een ander tool.



1. Bestudeer het LRS.

2. Maak in je database relaties aan die dit LRS correct implementeren. Leg echter nog geen DRI-constraints aan, of verwijder ze als je ze al hebt.
3. Maak een after trigger voor DELETE op Customer die voorkomt dat er customers verwijderd kunnen worden die nog orders hebben. Test de trigger.

4. Bouw een (after) trigger voor INSERT die checkt of bij alle records die toegevoegd worden aan Orders een customer bestaat in Customer.

   Als er voor één van de toegevoegde orders geen customer bestaat moeten alle records weer verwijderd worden door de transactie ongedaan te maken, en een foutmelding gegenereerd worden via RAISERROR. (Bedenk dat via het INSERT...SELECT-statement meer dan één record tegelijk ingevoerd kan worden; de trigger wordt dan toch maar één keer afgevuurd!)

   Hints
   Bewaar het aantal rijen dat geïnsert werd in een locale variable. Dit gaat het makkelijkst door de globale variabele @@ROWCOUNT in het begin van de trigger uit te lezen.

   Als er geen records werden geïnsert (bijvoorbeeld omdat er niet voldaan werd aan een of andere WHERE-clause), spring je meteen uit de trigger met het RETURN-statement (dit is niet meer dan een eenvoudige optimalisatie).
   Anders: controleer middels een IF-statement of het het aantal rijen dat je krijgt door INSERTED te joinen met Customer op de kolom CustomerID ongelijk is aan het aantal rijen dat geïnsert werd. Zo ja, dan moet je een foutmelding geven en de transactie terugdraaien. Test de trigger met goede en foute inserts!

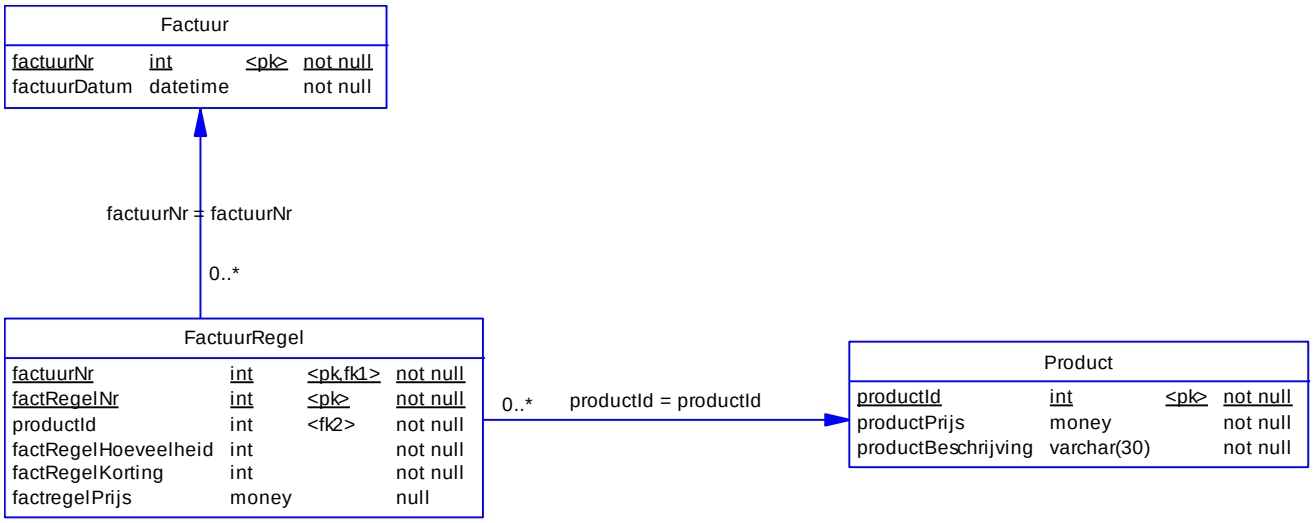
5. Maak een after trigger voor UPDATE op Orders die regelt dat er alleen toegestane statusovergangen worden ingevoerd.

   De toegestane statusovergangen staan in de volgende tabel:

	Registered	**Shipping**	**Ready**
Registered	**ja**	**ja**	**nee**
Shipping	**nee**	**ja**	**ja**
Ready	**nee**	**nee**	**ja**

Test de trigger met goede en foute updates!

6. Maak in je database relaties aan die de entiteiten OrderDetail en Product en de overige relationship correct implementeren. Gebruik hierbij declaratieve RI- constraints.
   Test de relaties.
   Bouw een after trigger voor INSERT op OrderDetail die regelt dat er altijd maar één record wordt geïnsert, en bovendien van het bijbehorende product de waarde van ProductsInStock vermindert met de ingevoerde DetailQuantity.
   Test de trigger met goede en foute inserts!
7. Geef een voorbeeld van een tabel met een INSERT AFTER trigger waarbij 0 records worden toegevoegd door de INSERT, en de trigger toch afgaat. (Het insert statement is hierbij van belang, niet de trigger).
8. Maak een trigger waarmee je kunt bepalen welk type statement (insert, update, delete) de trigger deed afvuren. Alleen als er geen rijen geraakt zijn hoeven we niet te weten welk type het was, in alle andere gevallen wel.
9. Een tabel tblBloodbankDonations heeft een not null column don_DonationDate van type datetime. Deze datum mag niet vóór de waarde liggen van de column prs_FirstTimeDonorDate van de betreffende donor in de tabel tblBloodBankPersons waarnaar tblBloodbankDonations verwijst. Regel dit met after triggers. (Eerder moest je dit doen met een stored procedure).
10. In triggers wordt vaak het statement
    IF UPDATE(<column>)
    gebruikt.
    Bestudeer dit statement in [JORGENSEN] of in Books OnLine, en maak een voorbeeld waarbij het gebruikt wordt in een trigger.
11. Maak een trigger die bij een insert checkt of een kolom wel een not-null waarde heeft. Waarom is dat geen zinvolle trigger?



12. Gegeven het volgende SQL Server schema, met voor de hand liggende tabellen en kolommen: De tabel FACTUURREGEL bevat een kolom FACTREGELPRIJS die automatisch wordt gevuld door triggers op de tabel FactuurRegel.
    Dat betekent o.a. dat bij een INSERT van FACTUURREGEL geen waarde voor FACTREGELPRIJS wordt meegegeven; een after trigger voor INSERT berekent de waarde uit FACTREGELHOEVEELHEID, de bijbehorende PRODUCTPRIJS en de FACTREGELKORTING. Geef deze after trigger. De trigger moet correct zijn voor multiple inserts.

Hint. Deze trigger moet een update doen op `FACTUURREGEL`. Gebruik hiervoor **NIET** de UPDATE…FROM van SQL SQL Server (omdat deze niet volgens de ANSI-standaard is, en erger nog, niet altijd goed werkt.). In de SET clause en in de WHERE clause zul je daarom moeten correleren met de tabel die in de UPDATE genoemd wordt, hier dus `FACTUURREGEL`.

Hieronder volgen de CREATE statements.

```
create table FACTUUR (
  FACTUURNR       int          not null,
  FACTUURDATUM     datetime        not null,
  constraint PK_FACTUUR primary key (FACTUURNR)
)
go
create table FACTUURREGEL (
  FACTUURNR       int          not null,
  FACTREGELNR      int           not null,
  PRODUCTID       int          not null,
  FACTREGELHOEVEELHEID int           not null,
  FACTREGELKORTING    int          not null,
  FACTREGELPRIJS     money         null,
```

```
  constraint PK_FACTUURREGEL primary key (FACTUURNR, FACTREGELNR)
)
go
create table PRODUCT (
  PRODUCTID      int         not null,
  PRODUCTPRIJS     money        not null,
  PRODUCTBESCHRIJVING varchar(30)     not null,
  constraint PK_PRODUCT primary key (PRODUCTID)
)
go
alter table FACTUURREGEL
  add constraint FK_FACTUURREGEL_FACTUUR foreign key (FACTUURNR)
   references FACTUUR (FACTUURNR)
go
alter table FACTUURREGEL
  add constraint FK_FACTUURREGEL_PRODUCT foreign key (PRODUCTID)
          references PRODUCT (PRODUCTID)
```

7 Thema Transactions

Leerdoelen

Zie Competenties en beoordelingscriteria DI-5, DI-9 in de studiehandleiding.

Theorieles en workshop

Onderwerpen:

- ACID
- Autocommit mode
- Expliciete transacties
- Geneste transacties

Workshop: oefeningen

Zelfstudie

- **Transactions**
  - Bestudeer uit [JORGENSEN] Chapter 47 t/m pagina 1054 (*Save Points)*.
  - Zie ook de tijdens de lessen gebruikte sheets.

Een transactie wordt vaak omschreven als een *logical unit of work*. Maar dat wil nog niet zeggen dat dat in SQL Server gegarandeerd is. De werkelijkheid is complexer. Je moet ook verstand hebben van transaction modes, transaction isolation levels, locking, nesting, error handling, enzovoorts (dit komt aan de orde bij Thema Concurrency).

Zo is het een wijd verbreid misverstand dat als binnen een transactie een constraint wordt geschonden, gedetecteerd door SQL Server, dat dan *automatisch* een rollback wordt gedaan. Helaas, dat is niet zo. Maak zelf maar eens een voorbeeldje waaruit dat blijkt. Error handling is daarom cruciaal!

8 Thema Indexing

Leerdoelen

Zie Competentie en beoordelingscriteria DI-8 in de studiehandleiding.

Theorieles en workshop

Onderwerpen:

- Storage structures
- Clustered en Nonclustered Indexes
- Table scans, Clustered Index scans, Index scans, Index seeks
- Performance tuning

Zelfstudie

- **Indexing.**
  - Lees [JORGENSEN] Chapter 45. Overslaan: *Identifying Key Queries*, *Specialty Indexes*.
  - Zie ook de tijdens de lessen gebruikte sheets.

.

8.1 Opdracht Execution plans

Doel

Doel van deze opdracht is om je vertrouwd te maken met verschillende execution plans van SQL Server. Je kunt hem nalezen in [JORGENSEN], chapter 45 vanaf pag 1024.

Inleiding

Gebruik AdventureWorks.

Je werkt met de tabel `Production.WorkOrder`. Deze tabel heeft 72591 rijen en 10 kolommen.

Definitie:

```
CREATE TABLE [Production].[WorkOrder](
        [WorkOrderID]  [int] IDENTITY(1,1)    NOT NULL,
        [ProductID]    [int]                  NOT NULL,
        [OrderQty]     [int]                  NOT NULL,
        [StockedQty] AS (isnull([OrderQty]-[ScrappedQty],(0))),
        [ScrappedQty]  [smallint]             NOT NULL,
        [StartDate]    [datetime]             NOT NULL,
        [EndDate]              [datetime]             NULL,
        [DueDate]              [datetime]             NOT NULL,
        [ScrapReasonID]        [smallint]             NULL,
        [ModifiedDate]         [datetime]             NOT NULL,
        CONSTRAINT [PK_WorkOrder_WorkOrderID] PRIMARY KEY CLUSTERED (
```

```
            [WorkOrderID] ASC)
)
```

De tabel heeft drie indexen, elk op één kolom (dit is op te maken uit de index name):

- IX_WorkOrder_ProductID (non-unique, non-clustered)
- IX_WorkOrder_ScrapReasonID (non-unique, non-clustered)
- PK_WorkOrder_WorkOrderID (primary key, clustered)

Statistics, Include execution plan, Clear buffers

Om na elke query zinvolle informatie te krijgen, en te zorgen dat die herhaalbaar is moet je het volgende doen.

Execute:

```
SET STATISTICS IO ON
```

Hierdoor krijg je informatie over de disk activity veroorzaakt door T-SQL statements.

Execute:

```
DBCC FREEPROCCACHE

DBCC DROPCLEANBUFFERS
```

Hierdoor worden buffers leeggemaakt. **Deze statements moet je uitvoeren vóór elk statement.**

Include het ***Actual Execution Plan*** (Management Studio, onder menu Query).

Overzichtstabel aanmaken

Noteer in Word je bevindingen na executie van de queries in de volgende paragraaf. Doe dat in een tabel.
Kolommen:
Query text
Execution plan (dwz Table scan, Clustered Index Seek, etc.)
Aantal Rijen
Logical Reads
Duur in ms
Aantal Rijen per ms
Opmerkingen.

Queries

```
SELECT *
FROM Production.WorkOrder

SELECT *
FROM Production.WorkOrder
WHERE WorkOrderID = 1234

SELECT *
FROM Production.WorkOrder
WHERE WorkOrderID BETWEEN 10000 AND 10010

SELECT *
FROM Production.WorkOrder
WHERE WorkOrderID BETWEEN 1 AND 72591
```

```
SELECT *
FROM Production.WorkOrder
WHERE StartDate = '2003-06-25'

SELECT WorkOrderID
FROM Production.WorkOrder
WHERE ProductID = 757

SELECT WorkOrderID
FROM Production.WorkOrder
WHERE ProductID + 1 = 757 + 1

SELECT WorkOrderID, StartDate
FROM Production.WorkOrder
WHERE ProductID = 757
```

Als een non-clustered index zelf elke kolom bevat die vereist is in de query, dan kan de optimizer besluiten om de query alleen met behulp van de non-clustered index uit te voeren. M.a.w., hij gaat dan niet meer naar de tabel zelf, die in de leaf level zit. Als dat zo is noemen we de index een *covering index* voor die query.

De vorige query had de kolom `StartDate` nodig. Omdat die kolom niet in de non-clustered index zit was de optimizer genoodzaakt een *bookmark lookup* te gebruiken van de non-clustered index naar de clustered index. De query is dan trager dan de vorige, die alles kon vinden in de non-clustered index.

Executeer nu het volgende. De transactie wordt alleen maar gebruikt om de zaak weer gemakkelijk te kunnen herstellen met een ROLLBACK, verder is die niet essentieel.

```
BEGIN TRAN
DROP INDEX Production.WorkOrder.IX_WorkOrder_ProductID
CREATE INDEX IX_WorkOrder_ProductID
      ON Production.WorkOrder (productID)
      INCLUDE (StartDate)
```

Hierdoor wordt geforceerd dat de kolom StartDate wordt toegevoegd aan de leaf level van de non-clustered index. Nu is de index een covering index voor de query geworden, en kan de optimizer een execution plan kiezen met enkel een index seek voor:

```
SELECT WorkOrderID, StartDate
FROM Production.WorkOrder
WHERE ProductID = 757
```

Executeer een `ROLLBACK` als je deze optie onderzocht hebt.

8.2 Opdracht Indexen aanmaken

Gegeven de volgende tabel, met een script om de tabel te vullen met 200000 records. In de system database *tempdb* gaat dit het snelst.

```
CREATE TABLE PerformanceIssue (
   ID UNIQUEIDENTIFIER NOT NULL,
   Code INT NOT NULL,
   Status CHAR(1) NOT NULL
)
go
DECLARE @Counter INT
DECLARE @ID UNIQUEIDENTIFIER
DECLARE @Status CHAR(1)
```

```
SET @Counter = 1

WHILE @Counter <= 200000
 BEGIN
   SET @ID = NewID()-- Creates a unique value of type uniqueidentifier.
   SET @Status = CASE @Counter % 2
                   WHEN 0 THEN 'A'
                   WHEN 1 THEN 'B'
                 END

   INSERT INTO PerformanceIssue (ID, Code, Status) VALUES (@ID, @Counter,
@Status)
   SET @Counter = @Counter + 1
 END
go
```

Zet nu(en niet vóórdat je de inserts uitvoert!) in de Management Studio *Include Actual Execution Plan* aan (onder menu *Query*).

a) Zet de volgende index op de tabel:

```
CREATE UNIQUE CLUSTERED INDEX UN_CL_ID
  ON PerformanceIssue(ID)
```

Voer de volgende query uit, en bekijk het execution plan:

```
SELECT ID, Code, Status
FROM PerformanceIssue
WHERE ID = 'CFED69CB-A8BE-4461-8B94-000537771CCD'
```

(Of zoek op een ander ID dat op jouw laptop gegenereerd is als je dat liever doet, dat maakt niet uit).
Waarom wordt dit plan gebruikt?

b) Drop nu de index uit a):

```
DROP INDEX PerformanceIssue.UN_CL_ID
```

Maak daarna de volgende index aan:

```
CREATE UNIQUE NONCLUSTERED INDEX UN_NONCL_ID
  ON PerformanceIssue(ID)
```

En voer weer de query uit a) uit, en bekijk het execution plan:

```
SELECT ID, Code, Status
FROM PerformanceIssue
WHERE ID = 'CFED69CB-A8BE-4461-8B94-000537771CCD'
```

Waarom wordt dit plan gebruikt?

c) Laat de index uit b) nog even staan.

Voer de volgende query uit, en bekijk het execution plan:

```
SELECT ID, Code, Status
FROM PerformanceIssue
```

```
WHERE Status = 'A'
```

Waarom wordt dit plan gebruikt?

d) Drop nu de index uit b):

```
DROP INDEX PerformanceIssue.UN_NONCL_ID
```

Maak daarna de volgende index aan:

```
CREATE CLUSTERED INDEX CL_Status
  ON PerformanceIssue(Status)
```

En voer weer de query uit c) uit, en bekijk het execution plan:

```
SELECT ID, Code, Status
FROM PerformanceIssue
WHERE Status = 'A'
```

Waarom wordt dit plan gebruikt?

e) Drop de index uit d):

```
DROP INDEX PerformanceIssue.CL_Status
```

Maak daarna de volgende index aan:

```
CREATE NONCLUSTERED INDEX NONCL_Status
  ON PerformanceIssue(Status)
```

En voer weer de query uit d) uit, en bekijk het execution plan:

```
SELECT ID, Code, Status
FROM PerformanceIssue
WHERE Status = 'A'
```

Waarom wordt dit plan gebruikt?

f) Voer de volgende query uit, en bekijk het execution plan:

```
SELECT COUNT(*)
FROM PerformanceIssue
WHERE Status = 'A'
```

Waarom wordt dit plan gebruikt?

9 Thema Concurrency

Leerdoelen

Zie Competenties en beoordelingscriteria DI-5, DI-6, DI-9 in de studiehandleiding.

Theorieles en workshop

Onderwerpen:

- Locks en concurrency
- Concurrency problems
- Isolation levels
- Deadlocks
- Optimistic vs pessimisitic concurrency control.

Zelfstudie

- **Concurrency**
  - Bestudeer uit [JORGENSEN] Chapter 47, pag 1054 e.v. Overslaan: *Monitoring Locking and Blocking*. Van de Lock Modes alleen S-locks en X-locks. Overslaan: *Controlling Lock timeouts*, *Lock duration*, *Index-level Locking Restrictions*, *Snapshot Isolations*, *Using Locking Hints*, *Application Locks*, *Transaction-Log Architecture*.
  - Zie ook de tijdens de lessen gebruikte sheets.

9.1 Opdrachten concurrency

Naast de opdrachten die in de presentatie opgenomen zijn kun je onderstaande extra opdrachten maken.

Maak eerst de volgende twee tabellen aan in SQL Server:

```
CREATE TABLE Orders (
orderNr         INT         PRIMARY KEY,
product     VARCHAR(30)   NOT NULL,
aantalBesteld   INT         NOT NULL
)

GO

CREATE TABLE Voorraad (
product              VARCHAR(30) PRIMARY KEY,
aantalInVoorraad   INT        NOT NULL
)

INSERT Voorraad (product, aantalInVoorraad) VALUES ('iPhone', 40)
INSERT Voorraad (product, aantalInVoorraad) VALUES ('iPad', 80)
```

1) Twee clients van SQL Server voeren elk ongeveer tegelijkertijd een transactie uit waarbij de tabel **Voorraad** betrokken is, Transaction 1 respectievelijk Transaction 2 zoals hieronder gegeven met hun isolation levels.

Transaction T1 **REPEATABLE READ**	**Transaction T2** **READ COMMITTED**
BEGIN TRANSACTION UPDATE **Voorraad** SET **aantalInVoorraad** = **aantalInVoorraad** + **100** WHERE **aantalInVoorraad** < **100** SELECT * FROM **Voorraad** WHERE **aantalInVoorraad** < **100** COMMIT TRANSACTION	INSERT **Voorraad** (product,**aantalInVoorraad**) VALUES ('SmartTV', **1**)

De server voert deze transacties interleaved uit.
Geef één mogelijk interleave-scenario waaruit blijkt dat er een concurrency-probleem kan zijn.
Geef aan ***welke*** locks (S-locks of X-locks) SQL Server zet, en ***wanneer*** en ***hoelang***. Geef óók aan ***waarom*** dat gebeurt!

Geef dit aan in een tabel als de volgende:

Tijd	**Transaction T1**	**Transaction T2**
1		
2		
3		
4		

Etc.

2) Twee clients van SQL Server voeren elk ongeveer tegelijkertijd een expliciete transactie uit waarbij de tabellen Orders en Voorraad betrokken zijn, Transaction 1 respectievelijk Transaction 2 zoals hieronder gegeven.

Beide transacties moeten draaien onder isolation level ***read committed.*** Zorg hiervoor.

SQL Server voert de statements in deze transacties interleaved uit. Geef een interleave-scenario in de tijd dat resulteert in een ***deadlock***. Gebruik in dit scenario voor elke connectie transaction statements (keuze uit BEGIN TRAN, COMMIT TRAN, ROLLBACK TRAN), en DML-statements op de tabellen (keuze uit SELECT, UPDATE, DELETE, INSERT).

Geef ***precies*** aan **welke S-locks** of **X-locks** SQL Server zet, en **wanneer**. Geef óók aan ***waarom*** dat gebeurt.
Geef het scenario aan in een tabel als de volgende.

Tijd	**Transaction T1** **Read Committed**	**Transaction T2** **Read Committed**
1		
2		
3		
4		
5		
6		
7		

3) In deze opdracht gaan we bekijken of een reeds geimplementeerde business rule (‘Er mag maximaal 1 president bestaan’) ook werkt in een multi-user interleaved scenarion.

Hieronder is de stored procedure gegeven waarin deze business rule geimplementeerd is. Bedenk wel dat de stored procedure een oplossing van een opdracht uit week 2/3 betreft. Er ontbreekt van alles aan (zoals elementen van het stored procedure template, transactiestatements, ed).

Dat kun/moet je toevoegen als je de integrity problemen agv concurrency issues wilt oplossen. Maar dat gaan we nu in eerste instantie even niet doen.

Vooraf:
Deze tabel moet je even aanmaken in b.v. de temp database om deze oefening te kunnen doen.

```
create table  emp

( empno       numeric(4,0)   not null
```

```
, ename      varchar(8)    not null
, job        varchar(9)    not null
, born       date          not null
, hired      date          not null
, sgrade     numeric(2,0)  not null
, msal       numeric(7,2)  not null
, username   varchar(15)   not null
, deptno     numeric(2,0)  not null
);
```

Deze tabel is logischerwijze leeg aan het begin van de oefening

De business rule is in onderstaande store procedure geimplementeerd:

```
CREATE PROCEDURE dbo.sp_InsertNewPresident
 (@empno numeric(4),
  @ename varchar(8),
  @job varchar(9),
  @born date,
  @hired date,
  @sgrade numeric(2,0),
  @msal numeric(7,2),
  @username varchar(15),
  @deptno numeric(2,0)
 )
AS
BEGIN
 BEGIN TRY
  --testen of er al een president bestaat
  IF EXISTS (SELECT '' FROM emp where job = 'PRESIDENT')
  THROW 50001, 'No more than one president allowed', 1
    INSERT INTO emp(empno,ename,job,born,hired,sgrade,msal,username,deptno)
  VALUES (@empno, @ename, @job, @born, @hired, @sgrade, @msal, @username,
@deptno);
 END TRY
 BEGIN CATCH
  ;THROW
 END CATCH
END

GO
```

De vraagstelling:
Kan het voorkomen dat er twee presidents in de database komen te staan wanneer je gebruik maakt van deze stored procedure voor het inserten van een nieuw record?

Laat zien of dat kan en neem zonodig maatregelen om te voorkomen dat er inderdaad integrity problemen kunnen ontstaan (gebruik **WAITFOR DELAY** op de juitse plek in de gegeven stored procedure om een switch naar een andere sessie te kunnen maken.
Plaats dus ergens het commando:

```
WAITFOR DELAY '00:00:15'
GO
```

Voor dan in één window in SSMS het volgende command uit:

```
EXEC  dbo.sp_InsertNewPresident @empno=1000, @ename = 'Hans', @job
='PRESIDENT', @born = '1957-12-22',
 @hired = '1992-01-01', @sgrade = 11, @msal = 15500,@username = 'HANS', @deptno =
10;
GO
```

Voer in het andere window (binnen 15 sec delay) op een ander tabpage in SSMS de volgende sql query uit:

```
EXEC  dbo.sp_InsertNewPresident @empno=2000, @ename = 'Chris', @job
='PRESIDENT', @born = '1959-06-19',
 @hired = '1992-01-01', @sgrade = 11, @msal = 7500,@username = 'SRISJE', @deptno =
10;
GO
```

Verklaar het gedrag van SQL Server.

Realiseer je dat SQL Server default draait in autocommit modus (elk statement is een transactie voor SQL Server tenzij jij aangeeft het zelf anders te willen: begin tran).

10 Thema Generatie van code

<<TO DO>>