DBSQL WERKBOEK MET ANTWOORDEN_

Opleiding HBO-ICT - Propedeuse

Mark Achten
Jorg Janssen

12 December 2024

HAN_UNIVERSITY
OF APPLIED SCIENCES

INHOUDSOPGAVE

1 INLEIDING

1.1 INTRO

Voor je ligt het werkboek voor het vak **DBSQL**, ofwel **DATSTQ03: Databases - Structured Query Language**. Het bevat oefeningen in de onderwerpen die je behandeld krijgt.

1.2 DOEL

SQL leer je niet van alleen theorie, je moet het vooral veel doen. Het doel van dit werkboek is om je zelfstandig (extra) te laten oefenen met SQL. Je vindt hierin diverse oefeningen van verschillend nivo. Je kunt vanaf de eerste les al oefeningen uit dit werkboek maken.

1.3 WERKWIJZE

Na iedere les heb je voldoende kennis om de opgaven te maken die in dit werkboek bij het betreffende thema staan. Oefeningen die als huiswerk zijn opgegeven zullen daarna zo goed als altijd ook in de klas worden behandeld. Ieder hoofdstuk heeft oefeningen over het betreffende thema en die hebben altijd eerst met de Muziekdatabase te maken. Je vindt die dan ook altijd als eerste paragraaf. Het advies is om per hoofdstuk en paragraaf één sql-bestand te maken met daarin jouw uitwerkingen. Bedenk dat er soms meerdere mogelijkheden zijn om tot hetzelfde eindresultaat te komen. Het kan dus voorkomen dat je een uitwerking gemaakt hebt die afwijkt van de antwoorden. Probeer dan zelf te bepalen of jouw uitwerking ook goed is. Let daarbij op dat als de resutaten van de query's overeenkomen het niet per se zo hoeft te zijn dat de query dan goed is. Bespreek dit bij twijfel altijd met je docent. Als je het niet vraagt, weet je ook niet of het goed is…

1.4 DATABASES

De oefeningen zijn onder andere gebaseerd op de muziekdatabase die je kent uit de lessen. Daarnaast gebruikt het werkboek nog een aantal andere databases. De schema's van deze databases en hoe je ze in je eigen SQL Server omgeving krijgt vind je op OnderwijsOnline.

1.5 SQL CONVENTIE

Bij deze uitwerkingen is gebruik gemaakt van de T-SQL conventie waarbij sleutelwoorden UPPERCASE worden geschreven. De layout van de query hoort daar ook bij. Deze is bedoeld om de uitwerking leesbaar te houden. Je hoeft je er niet aan te houden, maar net als bij andere vakken en programmeertalen geldt ook hier dat leesbaarheid en consistentie in code en query's een onderdeel van je professionaliteit zijn.

2 EENVOUDIGE QUERY'S

In dit hoofdstuk ga je oefenen met eenvoudige query's op één tabel. Ook ga je zien hoe je kunt filteren met de `WHERE`-clause, van eenvoudig tot wat moeiljker. Tenslotte komt ook het sorteren aan bod.

2.1 MUZIEKDATABASE

1. Geef alle informatie van alle stukken.

```
SELECT   *
FROM     Stuk;
```

2. Geef de naam en geboortedatum van alle componisten.

```
SELECT   naam, geboortedatum
FROM     Componist;
```

3. Geef stuknr, titel en genre van alle stukken.

```
SELECT   stuknr, titel, genrenaam
FROM     Stuk;
```

4. Geef alle informatie van klassieke stukken.

```
SELECT   *
FROM     Stuk
WHERE    genrenaam = 'klassiek';
```

5. Geef de namen en geboortedatums van die componisten die na 1920 geboren zijn. Tip: je kunt in veel programmeertalen gebruik van datum-functies, ook in SQL. Eén van die functies is de `YEAR()` functie die van een gegeven datum het jaar teruggeeft.

```
SELECT   naam, geboortedatum
FROM     Componist
WHERE    YEAR(geboortedatum) > 1920;

-- Of (maar bovenste optie heeft de voorkeur):
SELECT   naam, geboortedatum
FROM     Componist
WHERE    geboortedatum > '31-dec-1920';
-- Let hier op. Als je als datum '31-12-1920' opgeeft dan krijg je een foutmelding.
-- Analyseer de melding en probeer het te verklaren.

-- Beter is dan onderstaande oplossing:
SELECT   naam, geboortedatum
FROM     Componist
WHERE    geboortedatum >= '1-1-1921';
```

6. Geef alle informatie van speelstukken, ofwel stukken waarvan een niveau bekend is.

```
SELECT   *
FROM     Stuk
WHERE    niveaucode IS NOT NULL;
```

7. Geef alle informatie van klassieke stukken van niveaucode A met een speelduur van meer dan 5 minuten.

```
SELECT   *
FROM     Stuk
WHERE    niveaucode = 'A'
         AND genrenaam = 'klassiek'
         AND speelduur > 5;

 /*
 Hier hebben we drie rijfilters in de WHERE-clause. Als we die naast
 elkaar op één rij zouden schrijven wordt de regel minder goed
 leesbaar. Door ze in te laten springen is het in één oogopslag helder
 dat ze bij de WHERE-clause horen.
 */
```

8. Geef stuknr en titel van klassieke stukken waar het niveau niet bekend is.

```
SELECT   stuknr, titel
FROM     Stuk
WHERE    genrenaam = 'klassiek' AND niveaucode IS NULL;

/*
Hier is het nog leesbaar dat er twee criteria in de WHERE-clause staan en dus
dat ze op dezelfde regel geschreven kunnen worden.
*/
```

9. Geef de stukken die tussen 1950 en 1980 geschreven zijn. Schrijf de query op twee manieren.

```
SELECT   *
FROM     Stuk
WHERE    jaartal >= 1950 AND jaartal <= 1980;

-- Maar deze optie heeft echt de voorkeur!
SELECT   *
FROM     Stuk
WHERE    jaartal BETWEEN 1950 AND 1980;
```

10. Geef stuknummer en titel van alle jazz-stukken. Sorteer op titel.

```
SELECT   stuknr, titel
FROM     Stuk
WHERE    genrenaam = 'jazz'
ORDER    BY titel ASC;
```

11. Geef van alle speelstukken van na 1995 het stuknummer, het genre, de niveaucode en de speelduur. Sorteer het overzicht op speelduur (van groot naar klein) en bij gelijke speelduur op genre.

```
SELECT  stuknr, genrenaam, niveaucode, speelduur
FROM    Stuk
WHERE   niveaucode IS NOT NULL
        AND jaartal > 1995
ORDER   BY speelduur DESC, genrenaam;
```

12. Geef alle stukken van niveau A of B maar sowieso van origineel 1. Schrijf ook deze query op twee manieren.

```
-- Het gebruik van IN heeft de voorkeur.
SELECT  *
FROM    Stuk
WHERE   stuknrOrigineel = 1
        AND niveaucode IN ('A', 'B');

SELECT  *
FROM    Stuk
WHERE   stuknrOrigineel = 1
        AND (niveaucode = 'A' OR niveaucode = 'B');    -- LET HIER OP DE HAAKJES!!!
```

13. Geef alle info van componisten waarbij ergens in de naam ‘mozart’ staat.

```
SELECT  *
FROM    Componist
WHERE   naam LIKE '%mozart%';

-- SQL Server kent ook veel string-functies waar je gebruik van kunt maken:
SELECT  *
FROM    Componist
WHERE   CHARINDEX('mozart', naam) > 0;
```

14. Geef stuknummer, titel en genre van alle stukken die een bewerking zijn van stuk 5. Sorteer op titel.

```
SELECT  stuknr, titel, genrenaam
FROM    Stuk
WHERE   stuknrOrigineel = 5
ORDER   BY titel;
```

15. Geef de componistId, de naam en de geboortedatum van alle componisten die als docent verbonden zijn aan een muziekschool. Sorteer op geboortedatum (van jong naar oud).

```
SELECT  componistId, naam, geboortedatum
FROM    Componist
WHERE   schoolId IS NOT NULL
ORDER   BY geboortedatum DESC;
```

16. Bij het sorteren van het resultaat kun je ook getallen gebruiken. Bijvoorbeeld:

```
SELECT  titel, jaartal
FROM    Stuk
ORDER   BY jaartal;
```

Deze query levert hetzelfde op als:

```
SELECT  titel, jaartal
FROM    Stuk
ORDER   BY 2;
```

Hier wordt op de 2e kolom gesorteerd en dat is `jaartal`. Kun je voor- en nadelen bedenken van de ene manier en van de andere manier? En als je één manier zou moeten aanraden, welke zou dat dan zijn?

```
/*
Het gebruiken van de kolomnaam leest duidelijker, maar is meer typewerk.
Het gebruik van de positie is dus sneller qua typen maar leest dan
weer minder handig. Bij een kleine query is dat vaak geen probeem,
maar bij grote query's kan het wel een probleem zijn. Ook kan
de query later aangepast worden als er een kolom in de
SELECT-clause bij moet komen. Met het gebruik van een positie
in de ORDER BY kan het dus zijn dat er dan op een verkeerde
kolom wordt gesorteerd. Dat probleem heb je niet als je met kolom-
namen werkt.

Ofwel: het advies is hier om bij sortering altijd de kolomnamen
te gebruiken.
*/
```

2.2 PREMIER DATABASE

1. Vermeld het onderdeelnummer en de -beschrijving van alle onderdelen.

```
SELECT  PART_NUM, DESCRIPTION
FROM    PART;
```

2. Maak een lijst van alle rijen en kolommen voor de volledige REP-tabel.

```
SELECT  *
FROM    REP;
```

3. Maak een lijst van de namen van klanten met een kredietlimiet van ten minste $10.000.

```
SELECT  CUSTOMER_NAME
FROM    CUSTOMER
WHERE   CREDIT_LIMIT >= 10000;
```

4. Vermeld het bestelnummer voor elke bestelling geplaatst door klantnummer 148 op 23-10-2003. (Tip: om een datum in een voorwaarde in te voeren, voer je de dag van de maand, een koppelteken, de afkorting van drie tekens voor de maand, nog een koppelteken en het jaar in. De datum moet tussen enkele aanhalingstekens worden geplaatst. Om november in te voeren 15, 2003, bijvoorbeeld, typ je dus '15-nov-2003' in de WHERE-component).

```
SELECT  ORDER_NUM
FROM    ORDERS
WHERE   CUSTOMER_NUM = 148
        AND ORDER_DATE = '23-oct-2003';
```

5. Maak een lijst van nummer en naam van elke klant die wordt vertegenwoordigd door verkoper 20 of verkoper 65.

```
SELECT  CUSTOMER_NUM, CUSTOMER_NAME
FROM    CUSTOMER
WHERE   REP_NUM IN (20, 65);

-- Of deze, maar heeft niet de voorkeur.
SELECT  CUSTOMER_NUM, CUSTOMER_NAME
FROM    CUSTOMER
WHERE   REP_NUM = 20 OR REP_NUM = 65;
```

6. Vermeld het onderdeelnummer en de -beschrijving van elk onderdeel dat niet in artikelklasse HW valt.

```
SELECT  PART_NUM, DESCRIPTION
FROM    PART
WHERE   CLASS <> 'HW';  -- of: CLASS != 'HW';

-- Of:
SELECT  PART_NUM, DESCRIPTION
FROM    PART
WHERE   CLASS NOT IN ('HW');
```

7. Maak een lijst van het onderdeelnummer, de beschrijving en het aantal beschikbare eenheden voor elk onderdeel dat tussen de 10 en 25 eenheden op voorraad heeft.

```
SELECT  PART_NUM, DESCRIPTION, ON_HAND
FROM    PART
WHERE   ON_HAND BETWEEN 10 AND 25;
```

8. Maak een lijst van het onderdeelnummer, de onderdeelbeschrijving en de voorhanden waarde (voorhanden eenheden * eenheidsprijs) van elk onderdeel in artikelklasse AP. (Voorhanden waarde is in werkelijkheid beschikbare eenheden * kosten, maar er is geen COST-kolom in de PART-tabel). Wijs de naam ON_HAND_VALUE toe aan de berekening.

```
SELECT  PART_NUM, DESCRIPTION, (ON_HAND * PRICE) AS ON_HAND_VALUE
FROM    PART
WHERE   CLASS = 'HW';
```

9. Maak een lijst van het onderdeelnummer, de onderdeelbeschrijving en de voorhanden waarde voor elk onderdeel waarvan de voorhanden waarde minimaal $ 7.500 is. Wijs de naam ON_HAND_VALUE toe aan de berekening.

```
SELECT  PART_NUM, DESCRIPTION, (ON_HAND * PRICE) AS ON_HAND_VALUE
FROM    PART
WHERE   (ON_HAND * PRICE) >= 7500;
```

10. Gebruik de IN-operator om het onderdeelnummer en de onderdeelbeschrijving van elk onderdeel in itemklasse HW of SG weer te geven.

```
SELECT  PART_NUM, DESCRIPTION
FROM    PART
WHERE   CLASS IN ('HW', 'SG');
```

11. Zoek het nummer en de naam van elke klant wiens naam begint met de letter “D”.

```
SELECT  CUSTOMER_NUM, CUSTOMER_NAME
FROM    CUSTOMER
WHERE   CUSTOMER_NAME LIKE 'D%';
```

12. Maak een lijst van alle details over alle onderdelen. Sorteer de uitvoer op onderdeelbeschrijving.

```
SELECT  *
FROM    PART
ORDER   BY DESCRIPTION;
```

13. Maak een lijst van alle details van alle onderdelen. Sorteer de uitvoer op onderdeelnummer en artikelklasse.

```
SELECT  *
FROM    PART
ORDER   BY CLASS, PART_NUM;
```

3 AANMAKEN EN VULLEN TABELLEN

In dit hoofdstuk ga je oefenen met aanmaken en vullen van tabellen.

3.1 MUZIEKDATABASE

Zorg er bij deze opdrachten voor dat je dit uitvoert in de juiste database: de **MUZIEKDATABASE**.

1. Tabel **Orkest**.

   a. Schrijf het CREATE TABLE statement voor een tabel genaamd **Orkest** met de volgende kolommen:

      – **naam**, varchar(100), verplicht
      – **plaats**, varchar(50), verplicht
      – **url**, varchar(100), optioneel.

      Verder:

      – De primary key ligt op de kolom **naam**. Geef deze sleutel een betekenisvolle naam (bijvoorbeeld `PK_` gevolgd door de naam van de tabel).

```
CREATE TABLE Orkest
(
    naam varchar(100) NOT NULL,
    plaats varchar(50) NOT NULL,
    url varchar(100) NULL,

    CONSTRAINT PK_Orkest PRIMARY KEY (naam)
);
```

   b. Voeg twee rijen toe aan deze tabel, één orkest met een waarde voor **url** en één zonder. Bedenk zelf de namen van de orkesten of gebruik internet ter inspratie.

```
INSERT  INTO Orkest (naam, plaats, url)
VALUES  ('QHarmonie', 'Nijmegen', 'https://qharmony.nl');
INSERT  INTO Orkest (naam, plaats, url)
VALUES  ('Arnhems Promenade Orkest', 'Arnhem', NULL;);
```

   c. Kun je nu twee orkesten met dezelfde naam in dezelfde plaats toevoegen?

```
/*
Nee, dat wordt juist voorkomen door de primaire sleutel.
*/
```

   d. En kun je meerdere orkesten hebben in dezelfde plaats?

```
/*
Ja, dat kan wel. Als ze maar verschillende namen hebben.
*/
```

2. Tabel **Muzikant**.

   a. Schrijf het CREATE TABLE statement voor een tabel genaamd **Muzikant** met de volgende kolommen:

      – **lidnr**, numeric(7), verplicht
      – **naam**, varchar(100), verplicht
      – **instrument**, varchar(100), verplicht

Verder: De primaire sleutel ligt op de kolom **lidnr**. Geef deze sleutel een betekenisvolle naam.

```
CREATE TABLE Muzikant
(
    lidnr numeric(7) NOT NULL,
    naam varchar(100) NOT NULL,
    instrument varchar(100) NOT NULL,

    CONSTRAINT PK_Muzikant PRIMARY KEY (lidnr)
);
```

b. Kan dezelfde muzikant meerdere instrumenten bespelen volgens deze tabel?

```
/*
Nee. Dezelfde muzikant (met hetzelfde lidnummer) zou dan in meerdere rijen voorkomen
in deze tabel en dat kan niet vanwege de primary key.
*/
```

c. Voeg ook nu aantal zelf bedachte rijen toe aan deze tabel.

```
INSERT  INTO Muzikant (lidnr, naam, instrument)
VALUES  (1, 'Jan Janssen', 'Pauken');
INSERT  INTO Muzikant (lidnr, naam, instrument)
VALUES  (2, 'Janne Tikman', 'Viool');
INSERT  INTO Muzikant (lidnr, naam, instrument)
VALUES  (10, 'Yassin Alhamze', 'Viool');
```

3. Schrijf nu twee SQL-statements om de tabellen **Orkest** en **Muzikant** te verwijderen.

```
DROP TABLE Orkest;
DROP TABLE Muzikant;
```

4 REFERENTIËLE INTEGRITEIT

In dit hoofdstuk ga je oefenen met aanmaken en vullen van complexere tabellen: tabellen met foreign en alternate keys.

4.1 MUZIEKDATABASE

1. Welke constraints ken je nu allemaal?

```
/*
Primaire sleutel:                PRIMARY KEY
Alternatieve sleutel:            UNIQUE
Vreemde of verwijzende sleutel:  FOREIGN KEY
*/
```

2. Noem een aantal verschillen tussen een primaire sleutel en een alternatieve sleutel.

```
/*
Overeenkomsten:
- Beide over uniciteit: het bewaken van unieke waarden.
- Beide kunnen toegepast worden op een tabel, maar dat is niet verplicht.

Verschillen:
- Kolommen die de primaire sleutel vormen mogen niet optioneel zijn. Dat
mag wel bij een alternatieve sleutel.
- Per tabel mag er maar maximaal één primaire sleutel zijn. Dat geldt niet
voor alternatieve sleutels, daarvan mogen er meer zijn per tabel.
*/
```

3. Wat betekenen de volgende termen als het gaat om een `FOREIGN KEY`?

   a. Parent tabel

```
/*
Dat is de tabel waar naar verwezen wordt: REFERENCES <parent tabel> (...).
*/
```

   b. Child tabel

```
/*
Dat is de tabel van waaruit verwezen wordt.
*/
```

   c. Mag een kolom in de child tabel, die onderdeel is van een foreign key, optioneel zijn?

```
/*
Ja, dat mag. Die NULL waarden worden niet gecontroleerd in de parent tabel.
*/
```

4. Tabel **Concertgebouw**:

   a. Maak een nieuwe tabel genaamd **Concertgebouw** met de onderstaande kolommen:

   - **naam**, varchar(100), verplicht
   - **plaats**, varchar(50), verplicht
   - **url** varchar(100), optioneel.

De primaire sleutel van deze tabel ligt over de kolom **naam** en **plaats**. Geef deze PK een betekenisvolle naam. Verder mogen er geen concertgebouwen zijn die dezelfde url hebben. Geef ook deze constraint een betekenisvolle naam.

```
CREATE TABLE Concertgebouw
(
   naam varchar(100) NOT NULL,
   plaats varchar(50) NOT NULL,
   url varchar(100) NULL,

   CONSTRAINT PK_Concertgebouw
      PRIMARY KEY (naam, plaats),

   CONSTRAINT AK_Concertgebouw_url
      UNIQUE (url)
);
```

b. Kunnen er volgens deze tabel meerdere concertgebouwen zijn in dezelfde plaats?

```
/*
 Ja, er kunnen meerdere rijen in deze tabel staan waar de plaats hetzelfde is. Dat kan
 overigens alleen als de naam van het gebouw niet hetzelfde is.
*/
```

c. Kunnen er volgens deze tabel concertgebouwen zijn met dezelfde naam maar in verschillende plaatsen?

```
/*
Ja, dat kan ook. De uniciteit wordt bepaald door de combinatie van naam en plaats.
*/
```

d. Kunnen er meerdere gebouwen in deze tabel staan zonder url?

```
/*
Nee, dat kan niet. Probeer maar:

INSERT INTO Concertgebouw (naam, plaats, url)
VALUES ('De Vereeniging', 'Nijmegen', NULL);
INSERT INTO Concertgebouw (naam, plaats, url)
VALUES ('Music Arnhem', 'Arnhem', NULL);
*/
```

e. Vul de tabel met minimaal twee rijen.

```
INSERT INTO Concertgebouw (naam, plaats, url)
VALUES ('De Vereeniging', 'Nijmegen', 'https://www.stadsschouwburgendevereeniging.nl');
INSERT INTO Concertgebouw (naam, plaats, url)
VALUES ('Music Arnhem', 'Arnhem', NULL);
```

5. Tabel **Zaal**:

a. Maak een nieuwe tabel aan genaamd **Zaal** met de onderstaande kolommen:

- **naam**, varchar(50), verplicht. (naam van de zaal)
- **gebouw**, varchar(100), verplicht. (naam van het concertgebouw)
- **plaats**, varchar(50), verplicht.
- **capaciteit**, numeric(4), verplicht. (hoeveel bezoekers kunnen er in deze zaal) De primaire sleutel van deze tabel ligt over de kolommen **naam**, **gebouw** en **plaats**. De concertgebouwen moeten bestaande concertgebouwen zijn (ofwel: ze moeten voorkomen in de tabel **Concertgebouw**).

```
CREATE TABLE Zaal
(
   naam varchar(50) NOT NULL,
   gebouw varchar(100) NOT NULL,
   plaats varchar(50) NOT NULL,
   capaciteit numeric(4) NOT NULL,

   CONSTRAINT PK_Zaal PRIMARY KEY (naam, gebouw),

   CONSTRAINT FK_Zaal_Concertgebouw
      FOREIGN KEY (gebouw, plaats)
      REFERENCES Concertgebouw (naam, plaats)
);
```

b. Kunnen er zalen zijn met dezelfde naam maar in verschillende concertgebouwen?

```
/*
Ja, dat kan.
*/
```

c. Vul ook deze tabel, nu met minimaal vijf rijen.

```
INSERT   INTO Zaal (naam, gebouw, plaats, capaciteit)
VALUES   ('Kleine zaal', 'De Vereeniging', 'Nijmegen', 240);
INSERT   INTO Zaal (naam, gebouw, capaciteit)
VALUES   ('Theaterzaal', 'De Vereeniging', 'Nijmegen', 943);
INSERT   INTO Zaal (naam, gebouw, capaciteit)
VALUES   ('Parkzaal', 'Musis Arnhem', 'Arnhem', 1016);
INSERT   INTO Zaal (naam, gebouw, capaciteit)
VALUES   ('Muzenzaal', 'Musis Arnhem', 'Arnhem', 752);
INSERT   INTO Zaal (naam, gebouw, capaciteit)
VALUES   ('Parkfoyer', 'Musis Arnhem', 'Arnhem', 130);
```

d. Ben je bij de vorige vraag tegen een fout aangelopen? Zo ja, wat was die fout en verklaar waarom die optrad. Kreeg je geen fouten? Waar heb je dan bewust rekening mee gehouden?

```
/*
Zo ja: Dan is er zeer waarschijnlijk een FOREIGN KEY violation opgetreden en heb je
in deze child-tabel waarden geINSERT die niet voorkomen in de parent-tabel Concertgebouw.
Zo nee: Dan heb je dus waarschijnlijk rekening gehouden met de FOREIGN KEY constraint.
*/
```

6. Tabel **Uitvoering**:

a. Maak een nieuwe tabel aan genaamd **Uitvoering** met de onderstaande kolommen:

- **stuknr**, numeric(5), verplicht
- **zaal**, verplicht (kies hier zelf het beste datatype voor)
- **gebouw**, verplicht (kies hier zelf het beste datatype voor)
- **plaats**, verplicht (kies hier zelf het beste datatype voor)
- **datum**, verplicht (kies hier zelf het beste datatype voor)
- **bezoekers**, optioneel (kies hier zelf het beste datatype voor) De primaire sleutel van deze tabel ligt over de kolommen **stuknr**, **zaal**, **gebouw**, **plaats** en **datum**. De concertzalen moeten bestaan in tabel **Zaal** en de stuknummers moeten bestaande stukken zijn.

```
CREATE TABLE Uitvoering
(
   stuknr numeric(5) NOT NULL,
   zaal varchar(50) NOT NULL,
   gebouw varchar(100) NOT NULL,
   plaats varchar(50) NOT NULL,
   datum datetime NOT NULL,
   bezoekers numeric(4) NULL,
```

```
   CONSTRAINT PK_Uitvoering
      PRIMARY KEY (stuknr, zaal, gebouw, plaats, datum),

   CONSTRAINT FK_Uitvoering_Concertzaal
      FOREIGN KEY (zaal, gebouw, plaats)
      REFERENCES Zaal (naam, gebouw, plaats),

   CONSTRAINT FK_Uitvoering_Stuk
      FOREIGN KEY (stuknr)
      REFERENCES Stuk (stuknr)

);
```

b. Misschien heb je al bedacht dat het aantal bezoekers niet altijd al ingevuld kan worden (daarom is die ook optioneel). Want als een uitvoering nog moet komen dan weet je nog niet hoeveel bezoekers er zullen zijn. Er is nog zo'n regel die te maken heeft met het aantal bezoekers. Weet jij welke? Voorlopig mag je dit negeren, we zien later in deze course hoe we dit beter kunnen bewaken.

```
/*
Het aantal bezoekers moet groter of gelijk zijn aan 0 en kan niet meer zijn dan
de capaciteit van de zaal.
*/
```

c. Voeg aan deze tabel minimaal drie rijen toe.

```
INSERT   INTO Uitvoering (stuknr, concertzaal, gebouw, datum, bezoekers)
VALUES   (1, 'Kleine zaal', 'De Vereeniging', 'Nijmegen', '10-sep-2024 20:00', NULL);
INSERT   INTO Uitvoering (stuknr, concertzaal, gebouw, datum, bezoekers)
VALUES   (2, 'Kleine zaal', 'De Vereeniging', 'Nijmegen', '11-sep-2022 20:00', 134);
INSERT   INTO Uitvoering (stuknr, concertzaal, gebouw, datum, bezoekers)
VALUES   (2, 'Muzenzaal', 'Musis Arnhem', 'Arnhem', '10-sep-2024 20:00', NULL);
```

7. Schrijf nu een drietal SQL-statements om de tabellen **Concertgebouw**, **Zaal** en **Uitvoering** te verwijderen.

```
/*
-- De enige juiste volgorde waarbij er geen fouten optreden.
DROP TABLE Uitvoering;
DROP TABLE Zaal;
DROP TABLE Concertgebouw;
*/
```

5 TOEVOEGEN, WIJZIGEN EN VERWIJDEREN DATA

In dit hoofdstuk ga je leren om data in tabellen bij te werken, te verwijderen of te kopieren naar andere tabellen.

5.1 MUZIEKDATABASE

1. Het blijkt dat stuknr 10 een bewerking is van stuk 2. Schrijf een UPDATE-query die dat in de tabel bijwerkt. Schrijf daarna een SELECT-query om te kijken of je update gelukt is.

```
UPDATE   Stuk
SET      stuknrOrigineel = 2
WHERE    stuknr = 10;

SELECT   *
FROM     Stuk
WHERE    stuknr = 10;
```

2. Voeg een nieuw genre toe aan de genre tabel: Rock. Controleer met een SELECT-query of de INSERT is gelukt.

```
INSERT   INTO Genre (genrenaam)
VALUES   ('Rock');

-- Ook nu weer ff testen.
SELECT   *
FROM     Genre;
```

3. Verander het genre van ‘Swinging Lina’ (stuknr 13) naar rock en het niveau naar vergevorderden (= C). Ook nu graag testen om de UPDATE te controleren.

```
UPDATE   Stuk
SET      genrenaam = 'Rock',
         niveaucode = 'C'
WHERE    stuknr = 13;

/*
Hier worden overigens twee kolommen bjigewerkt in één statement. Door ze dan op verschillende
regels te schrijven wordt het leesbaarder.
*/

-- Test:
SELECT   *
FROM     Stuk
WHERE    stuknr = 13;
```

4. De speelduur van Blue bird (stuknr 1) klopt niet. Haal deze speelduur maar weg. Ook hier weer testen.

```
UPDATE   Stuk
SET      speelduur = NULL
WHERE    stuknr = 1;

-- Waarom nu niet: SET speelduur IS NULL???
-- Het gebruik van IS NULL is alleen van toepassing bij het
-- vergelijken met een NULL. Niet bij het toekennen.

-- Test:
SELECT   *
FROM     Stuk
WHERE    stuknr = 1;
```

5. Verander het genre van ‘Swinging Lina’ (nr 13) terug naar jazz.

```
UPDATE   Stuk
SET      genrenaam = 'jazz'
WHERE    stuknr = 13;
```

6. Verwijder het eerder toegevoegde genre ‘Rock’.

```
DELETE
FROM    Genre
WHERE   genrenaam = 'Rock';
```

7. Selecteer alle klassieke stukken en kopieer die naar een nieuwe tabel **KlassiekeStukken**.

   a. Doe dit met één statement.

```
SELECT   *
INTO     KlassiekeStukken
FROM     Stuk
WHERE    genrenaam = 'klassiek';
```

   b. Heeft deze tabel exact dezelfde structuur als de tabel **Stuk**?

```
/*
Nee. Bij een SELECT INTO worden de constraints NIET meegenomen.
*/
```

   c. Verwijder de tabel **KlassiekeStukken** weer.

```
DROP TABLE KlassiekeStukken;
```

   d. Maak nu weer die tabel **KlassiekeStukken**, maar nu door in de `WHERE` ervoor te zorgen dat er geen enkele rij geretourneerd wordt. Je maakt dus een ‘kopie’ van de tabel zonder data.

```
SELECT   *
INTO     KlassiekeStukken
FROM     Stuk
WHERE    1 = 2;
```

   e. Vul nu die tabel met een SELECT-statement op **Stuk** met alleen de klassieke stukken.

```
INSERT   INTO KlassiekeStukken
SELECT   *
FROM     Stuk
WHERE    genrenaam = 'klassiek';
```

   f. Verwijder de tabel **KlassiekeStukken** weer.

```
DROP TABLE KlassiekeStukken;
```

8. Een moeilijke: Het Muziekpakhuis (id 3) is verhuisd naar ‘s Hertogenbosch en heet nu: Het Bosche Huus. Pas dit aan. HINT: Je moet nu proberen om die quote van ’s Hertogenbosch mee te nemen in de waarde van het veld **plaatsnaam**. SQL Server moet dus die quote niet zien als begin of eind van een stuk tekst, maar als onderdeel van een stuk tekst. Dat heet ’escaping’. Zoek op het internet op hoe je dat in SQL Server doet.

```
UPDATE   Muziekschool
SET      naam = 'Het Bosche Huus',
         plaatsnaam = '''s Hertogenbosch'     -- hier vindt dus de escape plaats...
WHERE    schoolId = 3;

SELECT   *
FROM     Muziekschool
WHERE    schoolId = 3;
```

5.2 OPENSCHOOL

1. Geef het SQL statement dat de examinator verwijdert voor cursus DB. Let op, niet de docent verwijderen!!

```
UPDATE  Cursus
SET     examinator = NULL
WHERE   code = 'DB';

-- Testen:
SELECT  *
FROM    Cursus
WHERE   code = 'DB';
```

2. Men wil graag het aantal uren dat voor een cursus nodig is verhogen met 10%. Hoe luidt het SQL statement waarmee dat in één keer mogelijk wordt gemaakt?

```
UPDATE  Cursus
SET     Uren = Uren * 1.1;

-- Testen:
SELECT  *
FROM    Cursus;
```

3. Beschrijf wat er moet gebeuren als men van de course DB een tweede examinator (namelijk DAT) wil vastleggen.

```
/*
Daar zijn verschillende opties voor, maar de beste is deze:

1. Maak een nieuwe tabel voor het vastleggen van de examinatoren per vak.
- Kolommen: code, examinator
- PK: code, examinator

2. Kopieer de bestaande examinatoren naar die nieuwe tabel:
-   INSERT  INTO <die nieuwe tabel>
    SELECT  code, examinator
    FROM    Cursus

3. Verwijder de kolom examinator uit de tabel Cursus.
- Dat leren we een andere keer.
*/
```

6 WIJZIGEN EN VERWIJDEREN VAN TABELLEN

In dit hoofdstuk ga je leren om de structuur van tabellen bij te werken. Je gaat oefenen met het aanmaken van kolommen in bestaande tabellen, het wijzigen van kolommen en je oefent met het verwijderen van kolommen en tabellen.

6.1 MUZIEKDATABASE

1. We willen graag de genre tabel uitbreiden met een kolom waarin we meer informatie over dat genre kunnen opnemen. Gebruik als kolomnaam **info**, type `varchar(255)`.

   a. Schrijf het SQL-statement om de kolom aan de tabel toe te voegen.

   ```
   ALTER TABLE Genre
      ADD info varchar(255) NULL;
   ```

   b. Maakt het hier uit of je NULL of NOT NULL gebruikt?

   ```
   /*
   Ja. Omdat er al data in de tabel zit kun je geen verplichte kolom toevoegen.
   */
   ```

   c. Voeg voor alle genres in de nieuwe kolom een link toe naar een (relevante) corresponderende wikipedia-pagina. Schrijf hiervoor dus een aantal UPDATE-statements om dat voor alle genres te doen. Zoek de genres op Wikipedia (https://wikipedia.org) op en gebruik die url in je query's.

   ```
   UPDATE    Genre
   SET       Info = 'https://nl.wikipedia.org/wiki/Klassieke_muziek'
   WHERE     genrenaam = 'klassiek';

   UPDATE    Genre
   SET       Info = 'https://nl.wikipedia.org/wiki/Jazz'
   WHERE     genrenaam = 'jazz';

   UPDATE    Genre
   SET       Info = 'https://nl.wikipedia.org/wiki/Popmuziek'
   WHERE     genrenaam = 'pop';
   ```

   d. Nu alle rijen een waarde hebben voor die nieuwe kolom mag de kolom verplicht worden gemaakt. Schrijf het SQL-statement daarvoor.

   ```
   ALTER TABLE Genre
      ALTER COLUMN info varchar(255) NOT NULL;
   ```

   e. Om te zorgen dat de info betrouwbaar is willen we er zeker van zijn dat er niet meerdere genres dezelfde info hebben. Schrijf een SQL-statement om dat te realiseren.

   ```
   ALTER TABLE Genre
      ADD CONSTRAINT UQ_Genre_Info UNIQUE (info);
   ```

   f. Verwijder de constraint en de kolom.

```
ALTER TABLE Genre
   DROP CONSTRAINT UQ_info;

ALTER TABLE Genre
   DROP COLUMN Info;
```

g. Maakt de volgorde van de DROP-statements hierboven nog uit?

```
/*
Ja. Je kunt de tabel niet droppen als er nog unique in of foreign key constraints naar die t
*/
```

2. De geboortedatum van een componist bevat nu ook de tijd. Dat is niet nodig. Verander het datatype van die kolom naar datatype `date`.

```
ALTER TABLE Componist
  ALTER COLUMN geboortedatum date NULL;
```

6.2 OPENSCHOOL

1. Geef het SQL statement dat de examinator verwijdert voor cursus DB. Let op, niet de docent verwijderen!!

```
UPDATE  Cursus
SET     examinator = NULL
WHERE   code = 'DB';

-- Testen:
SELECT  *
FROM    Cursus
WHERE   code = 'DB';
```

2. Men wil graag het aantal uren dat voor een cursus nodig is verhogen met 10%. Hoe luidt het SQL statement waarmee dat in één keer mogelijk wordt gemaakt?

```
UPDATE  Cursus
SET     Uren = Uren * 1.1;

-- Testen:
SELECT  *
FROM    Cursus;
```

3. Eerder heb je beschrreven wat er moet gebeuren als men van de course DB een tweede examinator (namelijk DAT) wil vastleggen. Schrijf nu de SQL-statements om dat voor elkaar te krijgen.

```
/*
1. Maak een nieuwe tabel voor het vastleggen van de examinatoren per vak.
- Kolommen: code, examinator
- PK: code, examinator
*/
SELECT  code, examinator
INTO    Examinator
FROM    Cursus
WHERE   examinator IS NOT NULL;

ALTER TABLE Examinator
    ALTER COLUMN examinator char(3) NOT NULL;

ALTER TABLE Examinator
    -- Uiteraard een PK
    ADD CONSTRAINT PK_Examinator
            PRIMARY KEY (code, examinator),
```

```
        -- Ook een FK naar Cursus
        CONSTRAINT FK_Examinator_Cursus
            FOREIGN KEY (code)
            REFERENCES Cursus (code),
        -- En een FK naar Docent
        CONSTRAINT FK_Examinator_Docent
            FOREIGN KEY (examinator)
            REFERENCES Docent (acr);
-- Mag natuurlijk als losse ALTER TABLE statements worden geschreven.

-- Voeg de nieuwe examinator toe.
INSERT  INTO Examinator
VALUES  ('DB', 'DAT');

-- 2. Verwijder de kolom examinator uit de tabel Cursus. Daarvoor moet
-- eerst de FK naar Docent worden verwijderd.
ALTER TABLE Cursus
    DROP CONSTRAINT FK_Cursus_Docent;

ALTER TABLE Cursus
    DROP COLUMN examinator;
```

7 REFERENTIËLE INTEGRITEITSREGELS

In dit hoofdstuk ga je oefenen met wat de effecten kunnen zijn op andere tabellen als data bijgewerkt of verwijderd wordt.

7.1 MUZIEKDATABASE

1. Beschrijf in eigen woorden wat het eigenlijk inhoudt: een relatie tussen twee tabellen.

```
/*
Een relatie tussen twee tabellen houdt in dat de gegevens in de ene tabel op
een bepaalde manier verbonden zijn met de gegevens in een andere tabel.
En dan specifiek in kolommen die over hetzelfde gaan, zoals bijvoorbeeld het
stuknr. Het stuknr wordt in tabel Bezettinsgregel gebruikt. Dat stuknr
kun je dan weer opzoeken in de tabel Stuk om te weten om welk
stuk het precies gaat.
*/
```

2. Een relatie tussen tabellen implementeer je bij een `FOREIGN KEY` constraint.

   a. Op welk van de twee tabellen implementeer je de constraint? De parent-tabel of de child-tabel?

```
/*
Even ter verduidelijking: de parent-tabel is de tabel waarnaar verwezen wordt, de
child-tabel is de tabel van waaruit verwezen wordt. Dus in het voorbeeld
van tabellen Stuk en Bezettingsregel:
- parent: Stuk
- child: Bezettingsregel
Ofwel: de FOREIGN KEY constraint wordt op de child-tabel geïmplementeerd!
*/
```

   b. Waarom daar?

```
/*
De data in de child-tabel MOET vóórkomen in de parent-tabel. Wat in de parent-tabel
staat hoeft niet per se altijd voor te komen in de child-tabel.
Dus de controle MOET plaatsvinden op de child-tabel.
*/
```

   c. Als er een foreign key constraint wordt gemaakt, moet dan iedere waarde in de parent-kolom voorkomen in de child-kolom?

```
/*
Nee, dat hoeft dus niet per se. Het betekent alleen dat een waarde
in de child-kolom voor moet komen in de parent-kolom.
*/
```

   d. Als er een foreign key constraint wordt gemaakt, hoe vaak kan een waarde die in de parent-kolom staat dan voorkomen in de child-tabel?

```
/*
Die waarde kan dan of 0 of meer keer voorkomen. We noemen dat
dan een '0-op-veel' relatie.
*/
```

e. Welke acties op de parent-tabel zouden problemen op kunnen leveren als het gaat om referentiële integriteit? Geef bij elk van die acties een voorbeeld.

```
/*
Er zijn maar drie mogelijke acties: INSERT, UPDATE, DELETE:
1. INSERT: een INSERT van een waarde in de parent-tabel levert geen
problemen. Want die nieuwe waarde in de parent komt (nog) niet voor
in de child-tabel. Dat mag altijd.
2. UPDATE: een update van een waarde in de parent-tabel kan een probleem
opleveren. Als die waarde ook in de child-kolom voorkomt en je verandert
die waarde in de parent-tabel, dan zou er dus een waarde in de child-kolom
kunnen staan die dan niet meer in de parent-kolom staat. En dat is een
probleem.
3. DELETE: een DELETE van een rij in de parent-tabel zou er ook toe kunnen
leiden dat er een child-rij is waar een waarde in staat die dan niet meer in
de parent-tabel voorkomt. En dat mag niet.
*/
```

f. Welke acties op de child-tabel zouden problemen op kunnen leveren als het gaat om referentiële integriteit? Geef bij elk van die acties een voorbeeld.

```
/*
Er zijn maar drie mogelijke acties: INSERT, UPDATE, DELETE:
1. INSERT: Een INSERT van een waarde in de child-kolom die niet
in de parent-kolom staat. Als er bijvoorbeeld een regel in de
tabel Bezettingsregel wordt toegevoegd met een stuknr dat niet
in de tabel Stuk staat.
2. UPDATE: Een update van een waarde in de child-kolom naar
een waarde die niet in de parent-kolom staat, mag niet. Bv:
Het bijwerken van het stuknr in een bestaande rij uit
Bezettingsregel naar een niet-bestaand stuknr.
3. DELETE: Een delete van een rij uit de child-tabel heeft
geen problemen tot gevolg.
*/
```

g. Vul nu, op basis van je antwoorden hierboven de laatste kolom in de volgende tabel verder in. De kolom **Maatregel** geeft aan welke maatregel toegepast dient te worden. Gebruik alleen de opties:

- nvt (niet van toepassing)
- RI regel
- FK constraint

Actie	Op tabel	Maatregel
INSERT	parent	
UPDATE	parent	
DELETE	parent	
INSERT	child	
UPDATE	child	
DELETE	child	

Uitwerking:

Actie	Op tabel	Maatregel?
INSERT	parent	nvt
UPDATE	parent	RI Regel
DELETE	parent	RI Regel
INSERT	child	FK constraint
UPDATE	child	FK Constraint
DELETE	child	nvt

3. Beschrijf van de onderstaande SQL-statements welke rijen gewijzigd danwel verwijderd worden en motiveer je bevindingen. TIP: gebruik het CREATE-script voor de Muziekdatabase om te zien hoe de `FOREIGN KEY` constraints zijn aangemaakt.

   a. UPDATE Niveau SET niveaucode = 'E' WHERE niveaucode = 'A';

   ```
   /*
   Tabel Niveau is in de relatie met tabel Stuk de parent-tabel. Als RI regel geldt
   bij deze relatie: ON UPDATE CASCADE. Een update van de parent leidt dan tot
   automatische updates van de child voor overeenkomstige rijen.
   Bij een standaard vulling van de Muziekdatabase zijn dat stuknrs: 12, 14 en 15.
   LET OP: Officieel moeten we natuurlijk ook kijken of niveaucode 'E' al
   bestaat als niveau. Want deze update zou dan kunnen leiden tot een
   'primary key constraint violation'. Dat is gelukkig niet zo.
   */
   ```

   b. DELETE FROM Niveau WHERE niveaucode = 'C';

   ```
   /*
   Tabel Niveau is in de relatie met tabel Stuk de parent-tabel. Als RI regel geldt
   bij deze relatie: ON DELETE NO ACTION. Een delete van een parent gaat niet door
   als er overeenkomstige rijen in de child-tabel staan.
   Bij een standaard vulling van de Muziekdatabase zijn er geen stukken
   met niveaucode 'C', dus het DELETE-statement op tabel Niveau gaat door.
   */
   ```

   c. DELETE FROM Niveau WHERE niveaucode = 'B';

   ```
   /*
   Tabel Niveau is in de relatie met tabel Stuk de parent-tabel. Als RI regel geldt
   bij deze relatie: ON DELETE NO ACTION. Een delete van een parent gaat niet door
   als er overeenkomstige rijen in de child-tabel staan.
   Bij een standaard vulling van de Muziekdatabase zijn er weldegelijk stukken
   met niveaucode 'B', dus het DELETE-statement op tabel Niveau gaat nu NIET door.
   */
   ```

   d. UPDATE Stuk SET niveaucode = 'F' WHERE niveaucode = 'B';

   ```
   /*
   Tabel Stuk is in de relatie met tabel Niveau de child-tabel. RI reges gelden
   nu niet. Een update van een child-kolom mag alleen als de nieuwe waarde voorkomt
   in de parent-kolom.
   Bij een standaard vulling van de Muziekdatabase is er in de tabel Niveau geen
   niveaucode 'F', dus het UPDATE-statement op tabel Stuk gaat NIET door.
   */
   ```

   e. UPDATE Stuk SET stuknr = 99 WHERE stuknr = 1;

   ```
   /*
   Tabel Stuk heeft een aantal relaties in welke het steeds de parent-tabel is als
   het gaat om de kolom stuknr. Er zijn twee child-tabellen: Bezettingsregel en Stuk zelf.
   Door die laatste relatie geldt dat een update van een parent-kolom NOOIT ge-cascade
   mag worden naar child-tabellen (ON UPDATE NO ACTION).
   Bij een standaard vulling van de Muziekdatabase is stuknr 1 weldegelijk in gebruik
   als origineel bij stukken 2 en 15. Doordat deze update dus niet ge-cascade wordt,
   wordt ook die andere relatie niet bijgewerkt. Ook al staat daar ON UPDATE CASCADE.
   */
   ```

   f. UPDATE Stuk SET stuknr = 99 WHERE stuknr = 2;

   ```
   /*
   Tabel Stuk heeft een aantal relaties in welke het steeds de parent-tabel is als
   het gaat om de kolom stuknr. Er zijn twee child-tabellen: Bezettingsregel en Stuk zelf.
   Door die laatste relatie geldt dat een update van een parent-kolom NOOIT ge-cascade
   mag worden naar child-tabellen (ON UPDATE NO ACTION).
   ```

```
Bij een standaard vulling van de Muziekdatabase is stuknr 2 niet in gebruik
als origineel. Daardoor kan de update nu weldegelijk doorgaan.
*/
```

8 DEFINITIE VAN CONSTRAINTS

In dit hoofdstuk ga je oefenen met het bedenken en implementeren van CHECK-constraints.

8.1 MUZIEKDATABASE

1. In de Muziekdatabase is al een groot aantal constraints aanwezig. Maar ook nog niet. Voor deze vraag mag je uitgaan van de oorspronkelijke database zoals je die in les 1 hebt aangemaakt.

   a. Probeer zoveel mogelijk, nog niet geïmplementeerde constraints, te ontdekken. Begin met ze in woorden te beschrijven en denk out-of-the-box. Geef iedere constraint een nummer.

   ```
   /*
   1. Stuk:
       a. jaartal mag niet groter zijn dan huidig jaar
       b. speelduur mag niet negatief zijn
       c. jaartal moet later in tijd zijn dan geboortedatum van componist
       d. jaartal van een bewerking moet later zijn dan origineel
       e. een stuk kan nooit een bewerking zijn van zichzelf
   2. Componist:
       a. geboortedatum mag niet in de toekomst liggen
   3. Bezettingsregel
       a. aantal moet groter zijn dan 0
   */
   ```

   b. Geef van deze constraints aan óf je ze kunt implementeren en als je denkt dat dat kan, geef dan de SQL-instructies daarvoor. Gebruik zinvolle namen.

   ```
   -- 1a. Jaartal niet groter dan huidig jaar
   ALTER TABLE Stuk
       ADD CONSTRAINT CK_jaartal CHECK (jaartal <= YEAR(GETDATE()));

   -- 1b: Speelduur niet negatief zijn
   ALTER TABLE Stuk
       ADD CONSTRAINT CK_jaartal CHECK (speelduur > 0);

   -- 1c: Deze is (nog) niet te implementeren omdat je gegevens uit twee
   --   tabellen nodig hebt.

   -- 1d: Deze is (nog) niet te implementeren omdat je gegevens uit twee
   --   tabellen nodig hebt.

   -- 2a: Geboortedatum van een componist mag niet in de toekomst liggen
   ALTER TABLE Componist
       ADD CONSTRAINT CK_gebdatum CHECK (geboortedatum < GETDATE());

   -- 3a: Aantal instrumenten bij Bezettingsregel moet groter zijn dan 0
   ALTER TABLE Bezettingsregel
       ADD CONSTRAINT CK_aantal CHECK (aantal > 0);
   ```

   c. Schrijf nu voor iedere bedachte constraint twee SQL-testquery's (update of insert, dat maakt niet uit): één die niet voldoet aan de constraint en één die wel voldoet. Ofwel: een negatieve testquery en een positieve testquery.

– TIP: Je hoeft bij een INSERT niet voor álle kolommen een waarde op te geven. Doe dat alleen voor de verplichte kolommen, dat scheelt wat typen.

```
-- Omdat je hier zowel UPDATEs als INSERTs kunt gebruiken kan jouw uitwerking
-- afwijken van deze.

-- 1a. Stuk (jaartal) niet groter dan huidig jaar
-- Negatief
INSERT  INTO Stuk (stuknr, componistId, titel, genrenaam, jaartal)
VALUES  (1001, 1, 'TEST', 'klassiek', 2034);
-- Positief
INSERT  INTO Stuk (stuknr, componistId, titel, genrenaam, jaartal, speelduur)
VALUES  (1001, 1, 'TEST', 'klassiek', 1900, 10);

-- 1b: Speelduur niet negatief
-- Negatief
INSERT  INTO Stuk (stuknr, componistId, titel, genrenaam, jaartal, speelduur)
VALUES  (1001, 1, 'TEST', 'klassiek', 1900, -10);
-- Positief
INSERT  INTO Stuk (stuknr, componistId, titel, genrenaam, jaartal, speelduur)
VALUES  (1001, 1, 'TEST', 'klassiek', 1900, 10);

-- 1c: jaartal moet later in tijd zijn dan geboortedatum van componist
-- Dit kan (nu) nog niet worden geïmplementeerd.

-- 1d: jaartal van een bewerking moet later zijn dan origineel
-- Dit kan (nu) nog niet worden geïmplementeerd.

-- 2a. Geboortedatum componist mag niet in de toekomst liggen
-- Negatief
INSERT  INTO Stuk (stuknr, componistId, titel, genrenaam, jaartal, speelduur)
VALUES  (1001, 1, 'TEST', 'klassiek', 1900, -10);
-- Positief
INSERT  INTO Componist (componistId, naam, geboortedatum)
VALUES  (100, 'TEST', '');

-- 3a:
-- Negatief:
UPDATE  Bezettingsregel
SET     aantal = -1     -- of 0
WHERE   stuknr = 2 AND instrumentnaam = 'drums';
-- Positief:
UPDATE  Bezettingsregel
SET     aantal = 10
WHERE   stuknr = 2 AND instrumentnaam = 'drums';
```

2. Geef aan hoe je de volgende beperkingsregel afdwingt in het ALTER TABLE statement van de tabel Stuk: een stuk is nooit een bewerking van zichzelf.

```
-- Dat moet gelden: stuknr <> stukNrOrigineel
ALTER TABLE Stuk
    ADD CONSTRAINT CK_geen_eigen_bewerking  -- let op de zinvolle naam
        CHECK (stuknr <> stukNrOrigineel)   -- of: stuknr != stukNrOrigineel
```

3. Gegeven is de volgende beperkingsregel: een bewerking is altijd gemaakt van een origineel, en dus nooit van een andere bewerking. Beschrijf zo precies mogelijk wat dit betekent voor de tabel Stuk. Je hoeft het SQL-statement NIET te geven!!!

```
/*
Als een stuk wordt toegevoegd (of bijgewerkt) waarbij stukNrOrigineel
niet NULL is, dan geldt dat het stuk met stukNr met dat stukNrOrigineel
```

```
zelf GEEN stukNrOrigineel mag hebben.
*/
```

9 JOIN QUERY'S

9.1 INNER JOINS

In dit hoofdstuk ga je oefenen met het schrijven van query's met een `INNER JOIN`.

9.1.1 MUZIEKDATABASE

1. Geef alle informatie van componisten en hun gerelateerde stukken. Schrijf de query op twee manieren (oude en nieuwe syntax).

```
-- Oude syntax.
SELECT  *
FROM    Stuk, Componist
WHERE   Componist.componistId = Stuk.componistId;

-- Nieuwe syntax.
SELECT  *
FROM    Componist
        INNER JOIN Stuk ON Componist.componistId = Stuk.componistId;
```

2. Geef van alle stukken met een niveaucode (de speelstukken) stuknr, titel en de omschrijving van het niveau. Schrijf de query op twee manieren.

```
-- Oude syntax.
SELECT  stuknr, titel, omschrijving
FROM    Stuk, Niveau
WHERE   Stuk.niveaucode = Niveau.niveaucode;

-- Nieuwe syntax.
SELECT  stuknr, titel, omschrijving
FROM    Stuk
        INNER JOIN Niveau ON Stuk.niveaucode = Niveau.niveaucode;
```

3. Geef een lijst met namen van componisten en de naam van de school waaraan ze verbonden zijn/waren. Schrijf deze query op twee manieren.

```
-- Oude syntax.
SELECT  Componist.naam, Muziekschool.naam
FROM    Componist, Muziekschool
WHERE   Componist.schoolId = Muziekschool.schoolId;

-- Nieuwe syntax.
SELECT  Componist.naam, Muziekschool.naam
FROM    Componist
        INNER JOIN Muziekschool ON Componist.schoolId = Muziekschool.schoolId;
```

4. Geef van alle jazz-stukken ouder dan 1950 het stuknr, de titel, jaar en de naam van de componist. Sorteer aflopend op jaartal.

```
SELECT  stuknr, titel, jaartal, naam
FROM    Stuk
        INNER JOIN Componist ON Componist.componistId = Stuk.componistId
WHERE   genrenaam = 'jazz' AND jaartal < 1950
ORDER   BY jaartal DESC;
```

5. Geef per klassiek stuk het stuknr, de titel, de naam van de componist en de naam van de muziekschool waar de componist aan verbonden is/was. Als een componist niet verbonden is aan een muziekschool dan hoeft die niet opgenomen te worden in het resultaat. LET OP: je hebt hier drie tabellen voor nodig.

```
-- Omdat de kolom `naam` zowel in de Componist als de Muziekschool tabel voorkomt
-- moet je SQL Server hier helpen en vertellen uit welke tabel de naam gehaald
-- moet worden.
SELECT  stuknr, titel, Componist.naam, Muziekschool.naam
FROM    Stuk
        INNER JOIN Componist ON Stuk.componistId = Componist.componistId
        INNER JOIN Muziekschool ON Componist.schoolId = Muziekschool.schoolId
WHERE   genrenaam = 'klassiek';
```

6. Geef een overzicht van alle bewerkingen die zijn gecomponeerd door componisten die aan een muziekschool zijn/waren verbonden. Geef stuknr en titel van deze bewerkingen.

```
SELECT  stuknr, titel
FROM    Stuk AS S
        INNER JOIN Componist AS C ON S.componistId = C.componistId
WHERE   stuknrOrigineel IS NOT NULL AND schoolId IS NOT NULL;
```

7. Geef van alle stukken het stuknummer en de titel met ' – ' en de componistnaam in één kolom genaamd **Stukinfo**. Sorteer op stuknr.

```
SELECT  stuknr, titel + ' - ' + naam AS Stukinfo
FROM    Stuk
        INNER JOIN Componist ON Stuk.componistId = Componist.componistId
ORDER   BY stuknr;

-- Maar beter is deze:
SELECT  stuknr, CONCAT(titel, ' - ', naam) AS Stukinfo
FROM    Stuk
        INNER JOIN Componist ON Stuk.componistId = Componist.componistId
ORDER   BY stuknr;
```

8. Geef van alle stukken het stuknr, titel, jaartal, de naam van de componist, het geboortejaar van de componist en het verschil tussen het jaar van schrijven en het jaar van geboorte.

```
SELECT  stuknr, titel, jaartal, naam, YEAR(geboortedatum), jaartal - YEAR(geboortedatum)
FROM    Stuk
        INNER JOIN Componist ON Stuk.componistId = Componist.componistId;
```

9. Geef een lijst van stukken (stuknr en titel) waar een piano in wordt gebruikt.

```
SELECT  S.stuknr, titel
FROM    Stuk AS S
        INNER JOIN Bezettingsregel AS B ON S.stuknr = B.stuknr
WHERE   B.instrumentnaam = 'piano';
```

10. Nogmaals een lijst van stukken (stuknr en titel) waar een saxofoon in wordt gebruikt.

```
SELECT  S.stuknr, titel
FROM    Stuk AS S
        INNER JOIN Bezettingsregel AS B ON S.stuknr = B.stuknr
WHERE   B.instrumentnaam = 'saxofoon';
```

    a. Wat valt op aan het resultaat?

```
/*
Stuk 2 komt twee maal voor.
*/
```

    b. Hoe komt dat?

```
/*
Stuk 2 gebruikt zowel een alt-saxofoon als een tenor-saxofoon. Er wordt niet
geffilterd op toonhogte.
*/
```

c. Welk sleutelwoord moet je toevoegen aan de query om alle resutaten maar eenmaal te zien?

```
SELECT  DISTINCT S.stuknr, titel        -- DISTINCT
FROM    Stuk AS S
        INNER JOIN Bezettingsregel AS B ON S.stuknr = B.stuknr
WHERE   B.instrumentnaam = 'saxofoon';
```

11. Geef een overzicht van alle titels van bewerkingen van stukken met hun originele titel. Toon daarbij ook de stuknr's van bewerking en origineel.
    - HINT: je moet hier een self-join gebruiken.
    - HINT: Bij het gebruik van een self-join (een join van een tabel met zichzelf) MOET je tabelaliassen gebruiken.

```
SELECT  Bewerking.stuknr AS StuknrBewerking,
        Bewerking.titel AS TitelBewerking,
        Origineel.stuknr AS StuknrOrigineel,
        Origineel.titel AS TitelOrigineel
FROM    Stuk AS Bewerking
        INNER JOIN Stuk AS Origineel ON Bewerking.stuknrOrigineel = Origineel.stuknr;
```

9.1.2 OPENSCHOOL DATABASE

1. Geef een lijst van mentoren en de studenten waarvan ze mentor zijn. Sorteer op naam mentor en daarbinnen op naam van de student.

```
SELECT  D.naam, S.naam
FROM    Docent AS D
        INNER JOIN Student AS S ON S.mentor = D.acr
ORDER   BY D.naam, S.naam;
```

2. Geef de studenten die ooit een onvoldoende voor een tentamen hebben gekregen. Geef naam van de student, naam van het vak, het cijfer en datum.

```
SELECT  S.naam, C.naam, T.cijfer, T.datum
FROM    Tentamen AS T
        INNER JOIN Cursus AS C ON C.code = T.cursus
        INNER JOIN Student AS S ON S.nr = T.student
WHERE   T.cijfer < 5.5;
```

3. Geef een lijst van cursussen waar de examinator ook een begeleidend docent is (voor een willekeurig vak). In de output graag geen dubbele cursussen.

```
SELECT  DISTINCT C.*
FROM    Cursus AS C
        INNER JOIN Begeleider AS B ON C.examinator = B.docent;
```

9.1.3 PREMIER DATABASE

1. Geef bestelnummer, datum en klantnaam voor elke bestelling. Sorteer op naam en datum van de klant.

```
SELECT  O.ORDER_NUM, ORDER_DATE, C.CUSTOMER_NAME
FROM    ORDERS AS O
        INNER JOIN CUSTOMER AS C ON O.CUSTOMER_NUM = C.CUSTOMER_NUM
ORDER   BY CUSTOMER_NAME, ORDER_DATE;
```

2. Maak een lijst van elke klant met hun vertegenwoordiger (hun namen: achternaam, voornaam). Noem die kolom **REP**. LET OP: er zitten spaties in de namen. Zoek op internet hoe je de spaties kunt verwijderen. HINT: zoek op `TRIM`.

```
SELECT  C.CUSTOMER_NAME, TRIM(R.LAST_NAME) + ', ' + TRIM(R.FIRST_NAME) AS 'REP'
FROM    CUSTOMER AS C
        INNER JOIN REP AS R ON C.REP_NUM = R.REP_NUM;
```

3. Vermeld voor elke bestelling het bestelnummer, de datum, het onderdeelnummer, de onderdeelbeschrijving en het aantal. Sorteer op een logische manier.

```
SELECT  O.ORDER_NUM, ORDER_DATE, P.PART_NUM, P.DESCRIPTION, OL.NUM_ORDERED
FROM    ORDERS AS O
        INNER JOIN ORDER_LINE AS OL ON O.ORDER_NUM = OL.ORDER_NUM
        INNER JOIN PART AS P ON P.PART_NUM = OL.PART_NUM
ORDER   BY O.ORDER_NUM, DESCRIPTION;
```

9.2 OUTER JOINS

In dit hoofdstuk ga je oefenen met het schrijven van query's met een `OUTER JOIN`.

9.2.1 MUZIEKDATABASE

1. Geef een lijst van alle stukken (stuknr en titel) met een omschrijving van het niveau. Geef ook die stukken waarvoor geen omschrijving gevonden kan worden.

```
SELECT  stuknr, titel, omschrijving
FROM    Stuk AS S
        LEFT JOIN Niveau AS N ON S.niveaucode = N.niveaucode
```

2. Geef nu een lijst met alle niveau omschrijvingen en de stuknr's.

```
SELECT  omschrijving, stuknr
FROM    Niveau AS N
        LEFT JOIN Stuk AS S ON S.niveaucode = N.niveaucode;
```

3. Pas de query uit de vorige aan zodat je die niveaus ziet waarvoor geen stukken geschreven zijn.

```
SELECT  omschrijving, stuknr
FROM    Niveau AS N
        LEFT JOIN Stuk AS S ON S.niveaucode = N.niveaucode
WHERE   S.stuknr IS NULL;
```

4. Geef een lijst van alle componisten met hun eventuele stukken. Geef naam van de componist met het nummer van het stuk en de titel. Houd ook rekening met componisten die geen stukken hebben geschreven.

```
SELECT  naam, stuknr, titel
FROM    Componist AS C
        LEFT JOIN Stuk AS S ON C.componistId = S.componistId
```

5. Zijn er componisten die geen stukken hebben geschreven? Pas de query uit de vorige vraag aan zodat je daar antwoord op krijgt.

```
SELECT  C.*
FROM    Componist AS C
        LEFT JOIN Stuk AS S ON C.componistId = S.componistId
WHERE   stuknr IS NULL;

-- Bij de standaardvulling van de Muziekdatabase zijn er inderdaad geen componisten.
-- Dat wil dus niet zeggen dat de query fout is. De query is zo goed!
```

6. Welke klassieke stukken hebben een onbekende bezetting? Geef stuknr en titel.

```
SELECT  S.stuknr, titel
FROM    Stuk as S
        LEFT JOIN Bezettingsregel AS B ON S.stuknr = B.stuknr
WHERE   genrenaam = 'klassiek' AND B.stuknr IS NULL;
```

7. Zijn er genres waarvoor geen stukken bestaan?

```
SELECT  *
FROM    Genre
        LEFT JOIN Stuk ON Genre.genrenaam = Stuk.genrenaam
WHERE   stuknr IS NULL;
```

8. Geef een lijst van alle genres met hun eventuele stukken (alleen stuknr en titel).

```
SELECT  Genre.genrenaam, stuknr, titel
FROM    Genre
        LEFT JOIN Stuk ON Stuk.genrenaam = Genre.genrenaam;
```

   a. Test deze query door een genre aan de **Genre** tabel toe te voegen en te kijken of deze dan in het resultaat verschijnt. Schrijf dus een INSERT statement om een genre toe te voegen en voer de eerste query nogmaals uit.

```
INSERT INTO Genre VALUES ('Prog');
```

   b. Als de query goed is, kun je het toegevoegde genre weer verwijderen. Schrijf daar een SQL-statement voor.

```
DELETE
FROM    Genre
WHERE   genrenaam = 'Prog';
```

9. Geef een unieke lijst met namen van alle componisten en genres waar ze stukken voor geschreven hebben. Sorteer op genre, dan op naam van de componist.

```
SELECT  DISTINCT Componist.naam, genrenaam
FROM    Componist
        LEFT JOIN Stuk ON Componist.componistId = Stuk.componistId
ORDER   BY Componist.naam, genrenaam;
```

10. Geef een lijst van stukken (stuknr en titel) waar geen bewerking voor is geschreven.

```
SELECT  Origineel.stuknr, Origineel.titel
FROM    Stuk AS Origineel
        LEFT JOIN Stuk AS Bewerking ON Bewerking.stuknrOrigineel = Origineel.stuknr
WHERE   Bewerking.stuknr IS NULL;
```

9.2.2 OPENSCHOOL DATABASE

1. Voor welke vakken geldt dat ze niet als voorkennis worden beschouwd voor andere vakken? Geef code en naam van de cursus, gesorteerd op naam.

```
SELECT  code, naam
FROM    Cursus AS C
        LEFT JOIN Voorkenniseis AS V ON C.code = V.voorkennis
WHERE   V.cursus IS NULL
ORDER   BY naam;
```

2. Geef een lijst van cursussen waar de examinator juist geen begeleidend docent is (voor een willekeurig vak).

```
SELECT  DISTINCT C.*
FROM    Cursus AS C
        LEFT JOIN Begeleider AS B ON C.examinator = B.docent
WHERE   B.docent IS NULL
        AND examinator IS NOT NULL; -- waarom moet deze er nu bij en eerder niet???
```

9.2.3 PREMIER DATABASE

1. Maak een lijst van de onderdelen die nooit zijn besteld.

```
SELECT  P.*
FROM    PART AS P
        LEFT JOIN ORDER_LINE AS OL ON P.PART_NUM = OL.PART_NUM
WHERE   OL.PART_NUM IS NULL;
```

2. Maak een lijst van verkopers zonder enige klant.

```
SELECT  R.*
FROM    REP AS R
        LEFT JOIN CUSTOMER AS C ON C.REP_NUM = R.REP_NUM
WHERE   C.REP_NUM IS NULL;
```

3. Zijn er klanten die nooit iets besteld hebben?

```
SELECT  R.*
FROM    CUSTOMER AS C
        LEFT JOIN ORDERS AS O ON C.CUSTOMER_NUM = O.CUSTOMER_NUM
WHERE   O.CUSTOMER_NUM IS NULL;
```

10 STATISTISCHE QUERY'S

10.1 DEEL 1

In dit hoofdstuk ga je onder andere oefenen met het schrijven van query's met een `GROUP BY`.

10.1.1 MUZIEKDATABASE

1. Wat is de langste speelduur van een stuk?

```
SELECT  MAX(speelduur)
FROM    Stuk;
```

2. Wat is de gemiddelde speelduur van klassieke stukken?

```
SELECT  AVG(speelduur)
FROM    Stuk
WHERE   genrenaam = 'klassiek';
```

3. Geef per genre waar een stuk voor geschreven is het aantal stukken. Sorteer op aantal aflopend.

```
SELECT  genrenaam, COUNT(*) AS Aantal
FROM    Stuk
GROUP   BY genrenaam
ORDER   BY Aantal DESC;
```

4. Geef per stuk het stuknr en het totaal aantal instrumenten.

```
SELECT  stuknr, SUM(aantal)
FROM    Bezettingsregel
GROUP   BY stuknr;
```

5. Hoeveel instrumenten spelen er in totaal mee bij stuk 2?

```
SELECT  stuknr, SUM(aantal)
FROM    Bezettingsregel
WHERE   stuknr = 2;
```

6. Geef per niveau waar een stuk voor geschreven is het aantal stukken. Sorteer op aantal aflopend.

```
SELECT  niveaucode, COUNT(*) AS Aantal
FROM    Stuk
GROUP   BY niveaucode
ORDER   BY Aantal DESC;
```

7. Hoeveel componisten zijn niet verbonden aan een muziekschool?

```
SELECT  COUNT(*)
FROM    Componist
WHERE   schoolId IS NULL;
```

8. Geef per muziekschool waar een componist aan verbonden is, het aantal componisten.

```
SELECT  schoolId, COUNT(*)
FROM    Componist
GROUP   BY schoolId;
```

9. Geef een lijst van namen van componisten en het aantal stukken dat hij/zij geschreven heeft.

```
SELECT  C.componistId, naam, COUNT(*) AS Aantal
FROM    Stuk AS S
        INNER JOIN Componist AS C ON S.componistId = C.componistId
GROUP   BY C.componistId, naam;
```

10. Geef de totale speelduur van alle stukken.

```
SELECT  SUM(speelduur)
FROM    Stuk;
```

11. Geef per jaar waarin stukken geschreven zijn, het aantal stukken. Sorteer oplopend op jaar.

```
SELECT  jaartal, COUNT(*)
FROM    Stuk
GROUP   BY jaartal
ORDER   BY jaartal;
```

12. Geef een overzicht met per muziekschool het aantal componisten. Geef naam van de school en het aantal componisten.

```
SELECT  M.naam, COUNT(*) AS Aantal
FROM    Muziekschool AS M
        INNER JOIN Componist AS C ON M.schoolId = C.schoolId
GROUP   BY M.schoolId, M.naam;

/*
De GROUP BY op M.naam is nodig omdat we die in de SELECT hebben staan en
de GROUP BY op schoolId is nodig om groepjes te kunnen maken van iedere
school. Als we alleen op naam zouden groeperen dan komen scholen met
dezelfde naam in verschillende plaatsen toch in hetzelfde groepje en
tellen die dan als dezelfde school. En dat is niet de bedoeling. */
```

13. Geef per niveau de omschrijving en het aantal stukken dat ervoor geschreven is.

```
SELECT  omschrijving, COUNT(*)
FROM    Niveau AS N
        INNER JOIN Stuk AS S ON S.niveaucode = N.niveaucode
GROUP   BY omschrijving;
```

14. Geef per genre het aantal componisten die er stukken voor geschreven hebben.

```
SELECT  genrenaam, COUNT(componistId) AS Aantal
FROM    Stuk
GROUP   BY genrenaam;
```

15. Geef een lijst van instrumenten met toonhoogten en hoevaak ze in totaal in stukken voorkomen.

```
SELECT  instrumentnaam, toonhoogte, COUNT(*) AS Aantal
FROM    Bezettingsregel
GROUP   BY instrumentnaam, toonhoogte;
```

16. Zoals de vorige, maar nu zonder de toonhoogte erbij.

```
SELECT  instrumentnaam, COUNT(DISTINCT stuknr) AS Aantal
FROM    Bezettingsregel
GROUP   BY instrumentnaam;
```

10.2 DEEL 2

In dit hoofdstuk ga je oefenen met het schrijven van query's met een `GROUP BY` en `HAVING`. Ook worden de query's moeilijker door toepassing van outer joins.

10.2.1 MUZIEKDATABASE

1. Welk genre heeft van alle stukken een gemiddelde speelduur van meer dan 5 minuten?

```
SELECT  genrenaam
FROM    Stuk
GROUP   BY genrenaam
HAVING  AVG(speelduur) > 5;
```

2. Welk niveau heeft van alle stukken een gemiddelde speelduur van meer dan 5 minuten? Geef de niveaucode.

```
SELECT  niveaucode
FROM    Stuk
GROUP   BY niveaucode
HAVING  AVG(speelduur) > 5;
```

3. Voor welk niveau zijn meer dan drie stukken geschreven? Geef omschrijving en het aantal.

```
SELECT  omschrijving, COUNT(*) AS Aantal
FROM    Stuk AS S
        INNER JOIN Niveau AS N ON S.niveaucode = N.niveaucode
GROUP   BY N.niveaucode, omschrijving
HAVING  COUNT(*) > 3;
```

4. Welke componist heeft twee of meer stukken geschreven? Geef naam van de componist en het aantal stukken.

```
SELECT  naam, COUNT(*) AS Aantal
FROM    Componist AS C
        INNER JOIN Stuk AS S ON S.componistId = C.componistId
GROUP   BY C.componistId, C.naam
HAVING  COUNT(*) >= 2;
```

5. Geef een lijst van alle genres en het aantal stukken dat er voor geschreven is. Geef dus ook die genres waarvoor geen stukken geschreven zijn.

```
SELECT  G.genrenaam, COUNT(stuknr) AS Aantal
FROM    Genre AS G
        LEFT JOIN Stuk AS S ON G.genrenaam = S.genrenaam
GROUP   BY G.genrenaam;
```

6. Geef een lijst van genres waar geen enkel stuk voor geschreven is. Schrijf deze query op zoveel mogelijk manieren.

```
SELECT  G.genrenaam
FROM    Genre AS G
        LEFT JOIN Stuk AS S ON G.genrenaam = S.genrenaam
GROUP   BY G.genrenaam
HAVING  COUNT(stuknr) = 0;

-- Of:
SELECT  G.genrenaam
FROM    Genre AS G
        LEFT JOIN Stuk AS S ON G.genrenaam = S.genrenaam
WHERE   S.genrenaam IS NULL;
```

7. Zijn er instrumenten die niet in een bezetting van een stuk voorkomen? Geef instrumentnaam en toonhoogte.

```
SELECT  I.instrumentnaam, I.toonhoogte
FROM    Instrument AS I
        LEFT JOIN Bezettingsregel AS B
            ON I.instrumentnaam = B.instrumentnaam
            AND I.toonhoogte = B.toonhoogte
WHERE   B.stuknr IS NULL;
```

8. Geef voor alle componisten de naam en het aantal klassieke stukken stukken dat hij/zij geschreven heeft met daarbij de gemiddelde speelduur.

```
SELECT  naam, COUNT(stuknr) AS Aantal, AVG(speelduur) AS Gemiddeld
FROM    Componist AS C
        LEFT JOIN Stuk AS S
            ON C.componistId = S.componistId
            AND genrenaam = 'klassiek'
GROUP   BY C.componistId, naam;

-- Als je de filter van de klassieke stukken in de WHERE zou plaatsen
-- wordt er teveel weggefilterd waardoor de groepering een verkeerd
-- resultaat geeft.
-- Je kunt ook bedenken: join iedere componist met zijn klassieke stukken!
```

9. Geef voor alle genres het totaal aantal instrumenten dat bij alle stukken van dat genre gebruikt zijn.

```
SELECT  G.genrenaam
FROM    Genre AS G
        LEFT JOIN Stuk AS S ON G.genrenaam = S.genrenaam
        LEFT JOIN Bezettingsregel AS B ON S.stuknr = B.stuknr
GROUP   BY G.genrenaam;

-- De LEFT JOIN met Bezettingsregel is hier nodig omdat er wellicht
-- stukken bestaan zonder bezetting. Daardoor zouden er genres weg
-- kunnen vallen. Probeer deze maar eens te veranderen naar een
-- INNER JOIN en kijk wat er gebeurt.
```

10. Geef een lijst van de genres en niveaucodes waarvoor stukken geschreven zijn, samen met het aantal.

```
SELECT  genrenaam, niveaucode, COUNT(*)
FROM    Stuk
GROUP   BY genrenaam, niveaucode;
```

11 SUBQUERY'S

In het hoofdstuk ga je subquery's leren gebruiken. Je zult zien dat een aantal opgaven ook met een outer joini opgelost kan worden. Gebruik echter de subquery tenzij anders vermeld staat.

11.1 MUZIEKDATABASE

1. Geef de componistId en naam van alle componisten die klassieke stukken hebben geschreven.

```
SELECT  componistId, naam
FROM    Componist
WHERE   componistId IN (
    SELECT  componistId
    FROM    Stuk
    WHERE   genrenaam = 'klassiek');
```

2. Geef de componistId en naam van alle componisten die geen klassieke stukken hebben geschreven.

```
SELECT  componistId, naam
FROM    Componist
WHERE   componistId NOT IN (
    SELECT  componistId
    FROM    Stuk
    WHERE   genrenaam = 'klassiek');
```

3. Voor welke genres is nog geen stuk geschreven?

```
SELECT  genrenaam
FROM    Genre
WHERE   genrenaam NOT IN (
    SELECT  genrenaam
    FROM    Stuk);
```

4. Geef het stuknummer en de titel van het meest recent gecomponeerde muziekstuk.

```
SELECT  stuknr, titel
FROM    Stuk
WHERE   jaartal = (
    SELECT  MAX(jaartal)
    FROM    Stuk);

SELECT  stuknr, titel
FROM    Stuk
WHERE   jaartal = (
    SELECT  TOP 1 jaartal
    FROM    Stuk
    ORDER   BY 1 DESC);
```

5. Geef de naam van de oudste componist. Schrijf de query op twee manieren.

```
SELECT  naam
FROM    Componist
WHERE   geboortedatum = (
    SELECT  MIN(geboortedatum)
    FROM    Componist);

SELECT  naam
FROM    Componist
WHERE   geboortedatum = (
    SELECT  TOP 1 geboortedatum
    FROM    Componist
    ORDER   BY 1 ASC);
```

6. Geef het stuknummer en de titel van het minst recente klassieke speelstuk. LET OP: een speelstuk is dus een stuk met een niveaucode.

```
SELECT  stuknr, titel
FROM    Stuk
WHERE   niveaucode IS NOT NULL          -- Dit moet er wel bij
        AND genrenaam = 'klassiek'      -- en deze ook!!!
        AND jaartal = (
            SELECT  MIN(jaartal)                -- Ga dus eerst op zoek naar het minst recente
            FROM    Stuk                        -- klassieke speelstuk
            WHERE   niveaucode IS NOT NULL
                    AND genrenaam = 'klassiek');
```

7. Geef die componisten die wel klassieke stukken maar geen jazz stukken hebben geschreven.

```
SELECT  *
FROM    Componist
WHERE   componistId IN (
        SELECT  componistId
        FROM    Stuk
        WHERE   genrenaam = 'klassiek')
    AND componistId NOT IN (
        SELECT  componistId
        FROM    Stuk
        WHERE   genrenaam = 'jazz');

/* Deze query heeft twee subqquery's als onderdelen van de hoofdquery. Om dit leesbaar
en herkenbaar te houden is er hier voor gekozen op een ander niveau het inspringen
toe te passen. */
```

8. Welke componisten hebben alleen maar klassieke stukken geschreven? Geef id en naam van de componist.

```
SELECT  componistId, naam               -- Geef id en naam
FROM    Componist                       -- van componisten
WHERE   componistId IN (                -- waarvoor geldt dat het id voorkomt in de
        SELECT  componistId                 -- lijst van componist id's
        FROM    Stuk                        -- in stuk
        WHERE   genrenaam = 'klassiek')     -- die een klassiek stuk hebben geschreven
    AND componistId NOT IN (            -- en waarvoor geldt dat het id NIET voorkomt in de
        SELECT  componistId                 -- lijst van componist id's
        FROM    Stuk                        -- in Stuk
        WHERE   genrenaam != 'klassiek');   -- die iets anders dan klassiek hebben geschreven
```

9. Geef de namen van de componisten die een klassiek stuk hebben geschreven voor ‘beginners’.

```
SELECT  naam
FROM    Componist
WHERE   componistId IN (
    SELECT  componistId
```

```
    FROM    Stuk
    WHERE   genrenaam = 'klassiek'
    AND     niveaucode = (
        SELECT  niveaucode
        FROM    Niveau
        WHERE   omschrijving = 'beginners'));
```

10. Geef die componist(en) die van alle componisten de laagste gemiddelde speelduur van hun stukken hebben.

```
SELECT  C.componistId, naam, AVG(speelduur) AS GemiddeldeSpeelduur
FROM    Stuk AS S
        INNER JOIN Componist AS C ON S.componistId = C.componistId
GROUP   BY C.componistId, naam
HAVING  AVG(speelduur) = (
    SELECT  TOP 1 AVG(speelduur)
    FROM    Stuk
    GROUP   BY componistId
    ORDER   BY 1 ASC);
```

11. Welk stuk heeft de meeste instrumenten in de bezetting? Geef stuknr, titel en aantal instrumenten.

```
SELECT  S.stuknr, titel, SUM(aantal) AS AantalInstrumenten
FROM    Stuk AS S
        INNER JOIN Bezettingsregel AS B ON S.stuknr = B.stuknr
GROUP   BY S.stuknr, titel
HAVING  SUM(aantal) = (
    SELECT  TOP 1 SUM(aantal)
    FROM    Bezettingsregel
    GROUP   BY stuknr
    ORDER   BY SUM(aantal) DESC);
```

12. Geef een lijst van stuknr en titel van stukken die geschreven zijn door componisten die verbonden zijn/waren aan een muziekschool uit Amsterdam.

```
SELECT  stuknr, titel
FROM    Stuk
WHERE   componistId IN (
    SELECT  componistId
    FROM    Componist
    WHERE   schoolId IN (
        SELECT  schoolId
        FROM    Muziekschool
        WHERE   plaatsnaam = 'Amsterdam'));

-- Of:
SELECT  stuknr, titel
FROM    Stuk
WHERE   componistId IN (
    SELECT  componistId
    FROM    Componist AS C
            INNER JOIN Muziekschool AS M ON C.schoolId = M.schoolId
    WHERE   plaatsnaam = 'Amsterdam');
```

13. Geef een lijst van componisten (naam) die minimaal één stuk hebben geschreven waar een piano in voorkomt.

```
SELECT  naam
FROM    Componist
WHERE   componistId IN (
    SELECT  componistId
    FROM    Stuk
    WHERE   stukNr IN (
```

```
        SELECT  stukNr
        FROM    Bezettingsregel
        WHERE   instrumentnaam = 'piano'));

-- Of:
SELECT  naam
FROM    Componist
WHERE   componistId IN (
    SELECT  componistId
    FROM    Stuk AS S
            INNER JOIN Bezettingsregel AS B ON S.stuknr = B.stuknr
    WHERE   instrumentnaam = 'piano'
    GROUP   BY stuknr, componistId
    HAVING  COUNT(*) >= 1)
```

14. In welk jaar of jaren zijn de meeste stukken geschreven? Geef het jaar of jaren met het aantal.

```
SELECT  jaartal, COUNT(*) AS Aantal
FROM    Stuk
GROUP   BY jaartal
HAVING  COUNT(*) = (
    SELECT  TOP 1 COUNT(*)
    FROM    Stuk
    GROUP   BY jaartal
    ORDER   BY 1 DESC);
```

15. Van welk genre zijn de meeste stukken geschreven? Geef het genre (of genres) met het aantal.

```
SELECT  genrenaam, COUNT(*) AS Aantal
FROM    Stuk
GROUP   BY genrenaam
HAVING  COUNT(*) = (
    SELECT  TOP 1 COUNT(*)
    FROM    Stuk
    GROUP   BY genrenaam
    ORDER   BY 1 DESC);
```

16. Kun je de vorige uitwerking zo aanpassen dat je nu alleen die genres te zien krijgt waarvoor minder stukken geschreven zijn dan de meest voorkomende genres?

```
SELECT  genrenaam, COUNT(*) AS Aantal
FROM    Stuk
GROUP   BY genrenaam
HAVING  COUNT(*) < (        -- Alleen de = veranderen naar een <
    SELECT  TOP 1 COUNT(*)
    FROM    Stuk
    GROUP   BY genrenaam
    ORDER   BY 1 DESC);
```

11.2 PREMIER DATABASE

1. Welke onderdelen zijn nooit besteld? Gebruik een subquery.

```
SELECT  *
FROM    PART
WHERE   PART_NUM NOT IN (
    SELECT  PART_NUM
    FROM    ORDER_LINE);
```

2. Vermeld de meest recente bestelling.

```
SELECT  *
FROM    ORDERS
WHERE   ORDER_DATE = (
    SELECT  MAX(ORDER_DATE)
    FROM    ORDERS);
```

3. Welke klanten hebben twee of meer bestellingen geplaatst?

```
SELECT  *
FROM    CUSTOMER
WHERE   CUSTOMER_NUM IN (
    SELECT  CUSTOMER_NUM
    FROM    ORDERS
    GROUP   BY CUSTOMER_NUM
    HAVING  COUNT(*) >= 2);
```

4. Maak een lijst van de klanten die meer dan eens op dezelfde dag hebben besteld. Vermeld de naam van de klant, de datum en het aantal geplaatste bestellingen. Gebruik de techniek die volgens jou het beste is.

```
-- Check de data eerst om te kijken wat we kunnen verwachten. Ik laat de conventie
-- hier even achterwege.
SELECT * FROM ORDERS ORDER BY CUSTOMER_NUM, ORDER_DATE; -- Alleen 608

SELECT  CUSTOMER_NAME, ORDER_DATE, COUNT(*) AS TOTAL_ORDERS
FROM    ORDERS AS O
        INNER JOIN CUSTOMER AS C ON C.CUSTOMER_NUM = O.CUSTOMER_NUM
GROUP   BY C.CUSTOMER_NUM, CUSTOMER_NAME, ORDER_DATE
HAVING  COUNT(*) > 1;
```

5. Welke klant(en) hebben het vaakst besteld van alle klanten? Geef de naam van de klant op en het aantal keren dat de klant iets heeft besteld. Sorteer daarop.

```
SELECT  C.CUSTOMER_NUM, C.CUSTOMER_NAME, COUNT(*) AS TIMES_ORDERED
FROM    CUSTOMER AS C
        INNER JOIN ORDERS AS O ON C.CUSTOMER_NUM = O.CUSTOMER_NUM
GROUP   BY C.CUSTOMER_NUM, C.CUSTOMER_NAME
HAVING  COUNT(*) = (
    SELECT  TOP 1 COUNT(*)
    FROM    ORDERS
    GROUP   BY CUSTOMER_NUM
    ORDER   BY 1 DESC)
ORDER   BY 2;
```

6. Welk onderdeel is het meest besteld?

```
SELECT  P.PART_NUM, SUM(NUM_ORDERED) AS TOTAL_NUMBER_ORDERED
FROM    PART AS P
        INNER JOIN ORDER_LINE AS OL ON OL.PART_NUM = P.PART_NUM
GROUP   BY P.PART_NUM
HAVING  SUM(NUM_ORDERED) = (
    SELECT  TOP 1 SUM(NUM_ORDERED)
    FROM    ORDER_LINE
    GROUP   BY PART_NUM
    ORDER   BY 1 DESC);
```

7. In welke stad hebben klanten het hoogste saldo? Geef staat, stad en dat saldo als TOTAL_BALANCE_COUNT.

```
SELECT  STATE, CITY, SUM(BALANCE) AS TOTAL_BALANCE_COUNT
FROM    CUSTOMER
GROUP   BY STATE, CITY
HAVING  SUM(BALANCE) = (
    SELECT  TOP 1 SUM(BALANCE)
```

```
FROM    CUSTOMER
GROUP   BY STATE, CITY
ORDER   BY 1 DESC);
```

12 USER DEFINED FUNCTIONS

12.1 ZONDER CHECK-CONSTRAINTS

12.1.1 MUZIEKDATABASE

1. Schrijf een user-defined function `UF_geef_geboortejaar_van_componist` die van een gegeven componistId het geboortejaar teruggeeft. De functie krijgt dus de componistId als input en retourneert een getal (het geboortejaar). Schrijf daarnaast ook een query om de functie te testen.

```
CREATE OR ALTER FUNCTION UF_geef_geboortejaar_van_componist
(
    @componistId numeric(4)
)
RETURNS numeric(4)
AS
BEGIN

    RETURN  (
        SELECT  YEAR(geboortedatum)
        FROM    Componist
        WHERE   componistId = @componistId
    );

END
GO

-- En testen:
SELECT  componistId, geboortedatum, dbo.UF_geef_geboortejaar_van_componist(componistId)
FROM    Componist;
```

2. Schrijf een user-defined function `UF_geef_school_componist` die voor een opgegeven componistId de naam van de muziekschool teruggeeft. Als de componist niet aan een muziekschool gekoppeld is, mag er een NULL terug gegeven worden. Schrijf daarnaast ook een query om de functie te testen.

```
CREATE OR ALTER FUNCTION UF_geef_school_componist
(
    @componistId numeric(4)
)
RETURNS nvarchar(30)
AS
BEGIN

    RETURN  (
        SELECT  M.naam
        FROM    Muziekschool AS M
                INNER JOIN Componist AS C ON M.schoolId = C.schoolId
        WHERE   componistId = @componistId;
    );

END
GO
```

```
SELECT  componistId, schoolId, dbo.UF_geef_school_componist(componistId)
FROM    Componist;
```

3. Schrijf een user-defined function `UF_geef_aantal_instrumenten_voor_stuk` die voor een opgegeven stuk het aantal instrumenten teruggeeft. Als voor een gegeven stuk geen bezetting bekend is, geef dan 0 terug. Schrijf de body van de functie met één statement. Schrijf daarnaast ook een query om de functie te testen.
   - HINT: Zoek op internet naar de functie `ISNULL()`. Gebruik die om 0 terug te geven bij een onbekende bezetting.

```
CREATE OR ALTER FUNCTION UF_geef_aantal_instrumenten_voor_stuk
(
    @stuknr numeric(5)
)
RETURNS smallint
AS
BEGIN

    RETURN  (
        SELECT  ISNULL(SUM(Aantal), 0)
        FROM    Bezettingsregel
        WHERE   stuknr = @stuknr
    );

END
GO

SELECT  stukNr, dbo.UF_geef_aantal_instrumenten_voor_stuk(stukNr)
FROM    Stuk;
```

4. Schrijf een user-defined function genaamd `UF_geef_info_van_bewerking` die voor een gegeven stuknr kijkt of het een bewerking is. Als het een bewerking is, dan retourneert de functie het stuknr en de titel als één geheel (bv 1 - 'Blue Bird'). Als het geen bewerking is, dan retourneert de functie de tekst 'Dit is een origineel'. Schrijf daarnaast ook een query om de functie te testen.

```
CREATE OR ALTER FUNCTION UF_geef_info_van_bewerking
(
    @stuknr numeric(5)
)
RETURNS nvarchar(30)
AS
BEGIN

    RETURN (
        SELECT  IIF(
                    stukNrOrigineel IS NOT NULL,
                    CONCAT(stukNrOrigineel, ' - ', titel),
                    'Dit is een origineel'
                )
        FROM    Stuk
        WHERE   stukNr = @stukNr;
    )

END
GO

-- En testen:
SELECT  stuknr, dbo.UF_geef_info_van_bewerking(stuknr)
FROM    Stuk;
```

5. **UITSMIJTER**: Een bewerking kan zelf ook weer bewerkt worden. Schrijf een UDF, die van een opgegeven stuknr het aller-origineelste stuknr teruggeeft. Dus stel, uitgaande van de oorspronkelijke

vulling van de database, dat stuk 15 geen bewerking is van stuk 1 maar van stuk 2. Stuk 2 blijft een bewerking van stuk 1.

a. Schrijf een UPDATE-statement die stuk 15 daarvoor bijwerkt.

```
UPDATE  Stuk
SET     stukNrOrigineel = 2
WHERE   stuknr = 15;
```

b. Schrijf nu de UDF zelf. Je krijgt een paar hints:

- Je kunt gebruik maken van de `IIF()` functie. Deze functie roep je aan in de SELECT en krijgt drie argumenten:
  1. Conditie (is dit stuk een bewerking?)
  2. Actie als conditie TRUE is (selecteer origineel van stuk)
  3. Actie als conditie FALSE is (selecteer dan stuknr)
- En in een UDF kun je diezelfde UDF weer gebruiken.

```
CREATE OR ALTER FUNCTION UF_geef_origineel_stuknr
(
    @stuknr numeric(5)
)
RETURNS numeric(5)
AS
BEGIN

    RETURN (
        SELECT  IIF(
                    stukNrOrigineel IS NOT NULL,                    -- de conditie
                    dbo.UF_geef_origineel_stuknr(stukNrOrigineel),  -- conditie = TRUE
                    stuknr                                          -- conditie = FALSE
                )
        FROM    Stuk
        WHERE   stuknr = @stuknr
    );

END
GO

SELECT  *, dbo.UF_geef_origineel_stuknr(stuknr)
FROM    Stuk;
```

12.2 MET CHECK-CONSTRAINTS

We gaan nu user-defined functions in combinatie met CHECK-constraints gebruiken om complexere business rules te implementeren.

12.2.1 MUZIEKDATABASE

1. Als eerste de regel dat het jaartal van een stuk later in tijd moet zijn dan het geboortejaar van de componist.

   a. Controleer met een SQL statement of er nu stukken zijn die niet aan deze regel voldoen. Maak nog geen gebruik van de eerder geschreven functie (die uit het vorige hoofdstuk).

```
SELECT  *
FROM    Stuk AS S
        INNER JOIN Componist AS C ON s.componistId = S.componistId
WHERE   jaartal < YEAR(geboortedatum);
```

   b. Herschrijf deze query en gebruik nu de eerder geschreven functie.

```
SELECT  *
FROM    Stuk
WHERE   jaartal < dbo.UF_geef_geboortejaar_van_componist(componistId);
```

c. Schrijf nu een CHECK-constraint die dat voor je gaat doen.

```
-- Let er op dat je nu moet uitgaan van het positieve. Dus het geboortejaar
-- moet kleiner zijn dan het jaartal. Pas als dat geldt, dan mag het.
ALTER TABLE Stuk
    ADD CONSTRAINT CK_geboortejaar_van_componist_na_jaartal
        CHECK (dbo.UF_geef_geboortejaar_van_componist(componistId) < jaartal);
```

d. Schrijf nu twee SQL-statements waarbij je één stuk toevoegt aan de tabel `Stuk` dat niet voldoet aan de regel en één stuk toevoegt die wel voldoet.

```
-- Foutief:
INSERT  INTO Stuk
VALUES  (100, 1, 'Test', NULL, 'jazz', NULL, 10, 1900);

-- Correct:
INSERT  INTO Stuk
VALUES  (100, 1, 'Test', NULL, 'jazz', NULL, 10, 1920);
```

2. We willen afdwingen dat alle bewerkingen later geschreven zijn dan het origineel.

a. Controleer met een SQL statement of er nu stukken zijn die niet aan deze regel voldoen. HINT: hier heb je een self-join voor nodig.

```
SELECT  *
FROM    Stuk AS Originelen
        INNER JOIN Stuk as Bewerkingen ON Bewerkingen.stukNrOrigineel = Originelen.stuknr
WHERE   Originelen.jaartal > Bewerkingen.jaartal;
-- Of
WHERE   NOT (Originelen.jaartal < Bewerkingen.jaartal);
```

b. Schrijf een functie, die van een gegeven stuknr het jaartal teruggeeft.

```
CREATE OR ALTER FUNCTION UF_geef_jaartal_van_stuk
(
    @stukNr numeric(5)
)
RETURNS numeric(4)
AS
BEGIN

    RETURN (
        SELECT  jaartal
        FROM    Stuk
        WHERE   stuknr = @stuknr
    );

END
GO
```

c. Schrijf nu een SQL statement die controleer of er stukken zijn die niet voldoen aan de regel. Gebruik de zojuist geschreven functie.

```
SELECT  *
FROM    Stuk
WHERE   stukNrOrigineel IS NOT NULL
        AND dbo.UF_geef_jaartal_van_stuk(stukNrOrigineel) > jaartal;
```

d. Schrijf nu een CHECK-constraint die dit voor je gaat controleren.

```
ALTER TABLE Stuk
    ADD CONSTRAINT CK_jaartal_bewerking_na_jaartal_origineel
        CHECK (dbo.UF_geef_jaartal_van_stuk(stukNrOrigineel) > jaartal
                AND stukNrOrigineel IS NOT NULL);
```

e. Schrijf nu wederom twee query's om deze CHECK-constraint te testen. Eén met een foutief geval en één met een correct geval.

```
-- Foutieve bewerking van stuk 1:
INSERT  INTO Stuk
VALUES  (200, 1, 'Bewerking 1', 1, 'jazz', NULL, 5, 1950);

-- Correct:
INSERT  INTO Stuk
VALUES  (200, 1, 'Bewerking 1', 1, 'jazz', NULL, 5, 1960);
```

3. We gaan nu eens kijken wat dit (de regel uit de vorige vraag) voor originele stukken betekent. Stuk 14 is een bewerking van stuk 5. Stuk 14 is later in tijd geschreven dan stuk 5.

   a. Schrijf een UPDATE-statement voor het origineel stuk 5 en verander het jaartal naar één jaar later dan stuk 14. Ofwel: het origineel (stuknr 5) is nu geschreven in 1999.

```
UPDATE  Stuk
SET     jaartal = 1999
WHERE   stuknr = 5;
```

   b. Is de UPDATE gelukt? Hoe komt dat?

```
-- De update is gelukt. De CHECK-constraint wordt
-- NIET uitgevoerd voor orginele stukken!
```

   c. Hoe zou je dat oplossen?

```
/*
Een UDF die controleert of er bij een opgegeven stuknr bewerkingen
zijn die eerder in tijd zijn geschreven dan het opgegeven stuknr.
Idealiter zou dit een boolean retourneren (zoals in Java) maar
dat kent SQL Server niet. Er is wel een datatype BIT dat een 1
of een 0 kan zijn. Je zou een 1 voor een true kunnen gebruiken
en een 0 voor false.
Gebruik dan het resultaat van die UDF in een CHECK-constraint.
Als de UDF 0 teruggeeft als resultaat, dan moet de CHECK-constraint
de actie tegenhouden.
*/
```

12.2.2 OPENSCHOOL DATABASE

Schrijf nu zelf CHECK-constraints die gebruik maken van zelf gemaakte user-defined functions om de business rules te implementeren.

1. Een tentamen moet altijd later in tijd zijn dan de inschrijving.

   a. Schrijf een query om te controleren of er gevallen zijn die niet aan de regel voldoen.

```
SELECT  I.datum, T.datum
FROM    Tentamen AS T
        INNER JOIN Inschrijving AS I
            ON I.student = T.student
            AND I.cursus = T.cursus
WHERE   I.datum > T.datum;
```

   b. Schrijf nu een functie die de datum van de inschrijving van een student op een cursus terug-geeft.

```
CREATE OR ALTER FUNCTION UF_geef_datum_inschrijving
(
    @cursus varchar(4),
    @student int
)
RETURNS date
AS
BEGIN

    RETURN (
        SELECT  datum
        FROM    Inschrijving
        WHERE   cursus = @cursus AND student = @student
    );

END
```

c. Schrijf wederom een query om te controleren of er gevallen zijn die niet aan de regel voldoen maar gebruik nu de zojuist gemaakte functie.

```
-- De join is niet meer nodig omdat die info nu uit de functie komt.
SELECT  *
FROM    Tentamen
WHERE   dbo.UF_geef_datum_inschrijving(cursus, student) > datum;
```

d. Schrijf nu een CHECK-constraint om dit te laten controleren.

```
-- Omdat de query hiervoor op zoek gaat naar schendingen van de regel
-- geldt juist bij het toevoegen van de CHECK dat die conditie dus
-- NIET geldt.
ALTER TABLE Tentamen
    ADD CONSTRAINT CK_datum_tentamen_later_dan_inschrijving
        CHECK (NOT dbo.UF_geef_datum_inschrijving(cursus, student) > datum);

-- En die laatste regel had ook zo gekund:
    CHECK (dbo.UF_geef_datum_inschrijving(cursus, student) < datum);
```

e. Schrijf twee SQL-statements om een tentamen toe te voegen waarbij de datum van het tentamen voor de datum van inschrijving ligt en bij de tweede de datum van het tentamen erna ligt.

```
-- Bijvoorbeeld student 1 voor DB, ingeschreven op 2012-03-18.
-- Foutief:
INSERT  INTO Tentamen
VALUES  (1, 'DB', 3, '01-01-2012', 10);

-- Correct:
INSERT  INTO Tentamen
VALUES  (1, 'DB', 3, '01-01-2013', 10);
```

f. Wat zou er nu gebeuren als de datum van de inschrijving van een student op een cursus zou worden aangepast naar een datum die na het eerste tentamen van die cursist en die cursus zou liggen?

```
/*
Niets. De CHECK-constraint controleert alleen data in de Tentamen-tabel.
Ofwel: de regel zou dan geschonden worden en om dat op te lossen zou je
ook de Inschrijving tabel moeten controleren.
*/
```

2. Er mogen alleen cijfers voor tentamens worden geregistreerd als die student voor dat vak geen vrijstelling heeft.

   a. Schrijf een query die controleert of er rijen zijn waarvoor de regel niet geldt.

```
SELECT  *
FROM    Tentamen AS T
        INNER JOIN Inschrijving AS I
            ON I.student = T.student
            AND I.cursus = T.cursus
WHERE   I.vrijstelling = 'J';
```

b. Schrijf een UDF die, op basis van de student en de cursus, de code van de vrijstelling teruggeeft.

```
CREATE OR ALTER FUNCTION UF_geef_vrijstelling
(
    @student int,
    @cursus varchar(4)
)
RETURNS varchar(1)
AS
BEGIN

    RETURN (
        SELECT  vrijstelling
        FROM    Inschrijving
        WHERE   student = @student AND cursus = @cursus);

END
GO
```

c. Herschrijf de query uit a) en maak nu gebruik van de udf.

```
-- Ook hier vervalt de join omdat die info nu uit de function komt.
SELECT  *
FROM    Tentamen
WHERE   dbo.UF_geef_vrijstelling(student, cursus) = 'J';
```

d. Wijzig de tabel Tentamen en voeg daar een CHECK-constraint toe die gebruik maakt van die UDF.

```
ALTER TABLE Tentamen
    DROP CONSTRAINT IF EXISTS UF_geef_vrijstelling;

ALTER TABLE Tentamen
    ADD CONSTRAINT UF_geef_vrijstelling
        CHECK (NOT dbo.UF_geef_vrijstelling(student, cursus) = 'J');
        -- Of: CHECK (dbo.UF_geef_vrijstelling(student, cursus) != 'J');
```

e. Schrijf een positieve en negatieve testquery die de werking van de constraint test.

```
-- Positieve:
INSERT  INTO Tentamen
VALUES  (1, 'DB', 3, GETDATE(), 5); -- Let op juiste volgnummer

-- Negatieve:
INSERT  INTO Tentamen
VALUES  (2, 'DW', 10, GETDATE(), 5);
```

13 NORMALISEREN

13.1 OEFENINGEN

1. Gegeven onderstaande tabel. De primaire sleutel ligt over **StudentNr** en **CursusCode**. Verder zijn er de volgende aannames:
   - Een student doet dezelfde cursus maar één keer.
   - De naam van de student wordt gezien als één geheel.

Studentnr	Naam	Adres	Woonplaats	**Cursuscode**	Cursusnaam	Startdatum	Gebouw	Locatie
1234	Jan de Smet	Keistraat 30	Edam	63info	Informatica	1-9-2022	BBME	Rotterdam
5678	Willem Wever	Dorpstraat 1	Schiedam	62taal	Talen	1-9-2022	Atheneum	Den Haag
5678	Willem Wever	Dorpstraat 1	Schiedam	63info	Informatica	1-10-2022	Atheneum	Den Haag
9511	Willem Wever	Steenderens 12	Westervoort	63info	Informatica	1-11-2022	BBME	Rotterdam

   a. Wat zijn de afhankelijkheden?

```
-- (StudentNr, CursusCode) -> Startdatum, Gebouw
-- StudentNr -> Naam, Adres, Woonplaats
-- CursusCode -> Cursusnaam
-- Gebouw -> Locatie
```

   b. In welke normaalvorm staat deze tabel? Geef hierbij je motivatie en ook daarbij waarom een tabel niet voldoet aan de eerstvolgende normaalvorm. Dus als je vindt dat de tabel in 2NV staat, waarom vind je dat en waarom niet in 3NV. Stop in je motivatie ook een voorbeeld die dit onderschrijft.

```
/* Deze tabel staat in 1NV omdat alle waarden atomair zijn. De tabel staat
niet in 2NV omdat er niet-sleutel attributen zijn die afhankelijk zijn van
een deel van de primaire sleutel, zie bijvoorbeeld Naam. Die is alleen
afhankelijk van StudentNr en niet van (StudentNr, CursusCode). */
```

2. Gegeven onderstaande tabel met kandidaatsleutel op **Titel**, **Jaar** en **Acteur**. Er zijn films met dezelfde titel die in een ander jaar worden gemaakt. Deze films kunnen dezelfde regisseur of acteurs hebben.

Titel	**Jaar**	Regisseur	**Acteur**
Grease	1978	Randal Kleiser	Olivia Newton-John
Grease	1978	Randal Kleiser	John Travolta
Metro	1997	Thomas Carter	Eddie Murphy
Metro	1997	Thomas Carter	Kim Kiyori
Metro	1997	Thomas Carter	Michael Rapaport

   a. Welke afhankelijkheden zijn er?

```
-- (Titel, Jaar) -> Regisseur
```

   b. Waarom staat deze tabel niet in 2NV?

```
-- Er is dus een niet-sleutel attribuut afhankelijk van een deel
-- van de primaire sleutel (partitiële afhankelijkheid).
```

c. Zet de tabel om in twee tabellen in 2e normaalvorm. Geef de primary keys en eventuele foreign keys van die nieuwe tabellen.

FILM, PK (Titel, Jaar, Acteur):

Titel	**Jaar**	**Acteur**
Grease	1978	Olivia Newton-John
Grease	1978	John Travolta
Metro	1997	Eddie Murphy
Metro	1997	Kim Kiyori
Metro	1997	Michael Rapaport

REGISSEUR, PK (Titel, Jaar):

Titel	**Jaar**	Regisseur
Grease	1978	Randal Kleiser
Metro	1997	Thomas Carter

FK: FILM (Titel, Jaar) -> REGISSEUR (Titel, Jaar)

3. Een theater registreert vóór elk seizoen welke voorstellingen er op welke dagen en tijdstippen gegeven gaan worden, samen met de vastgestelde toegangsprijzen. Deze gegevens staan in een tabel PROGRAMMA. Zie hier beneden voor een voorbeeld. De primary key ligt op attributen **Datum** en **Tijd**. Een voorstelling wordt uniek geïdentificeerd door het voorstellingnummer (Vnr). De kolommen Normaal en Abonnement geven de prijzen weer.

PROGRAMMA:

Datum	**Tijd**	Vnr	Titel	Normaal	Abonnement
5-03-2008	20:15	24	Ballet Don Quichote	40	35
6-03-2008	14:00	25	Cowboy Billie Boem	15	12
6-03-2008	20:15	26	Lebbis en Jansen	25	20
7-03-2008	20:30	27	Aïda	50	45
8-03-2008	20:30	27	Aïda	50	45

a. Wat zijn de functionele afhankelijkheden?

```
-- Vnr -> Titel
-- Vnr -> Normaal
-- Vnr -> Abonnement
```

b. Waarom staat deze tabel in de 2e normaalvorm?

```
/*  4e afhankelijkheid is dus (Datum, Tijd) à Voorstellingsnr.
Oftewel, alle niet-sleutelattributen zijn functioneel (zowel direct
als indirect (transitief)) afhankelijk van de gehele primaire sleutel. */
```

c. Waarom staat deze tabel niet in de 3e normaalvorm?

```
/* Er zijn nog niet-sleutelattributen transitief (indirect) afhankelijk
van de primaire sleutel. Zie als voorbeeld de 3 gegeven functionele
afhankelijkheden zoals Voorstellingnr → Voorstellingtitel. */
```

d. Zet de tabel om in twee tabellen in 3e normaalvorm. Geef de primary keys en eventuele foreign keys van de nieuwe relaties.

PROGRAMMA: PK (Datum, Tijd)

Datum	**Tijd**	Vnr
5-03-2008	20:15	24
6-03-2008	14:00	25
6-03-2008	20:15	26
7-03-2008	20:30	27
8-03-2008	20:30	27

VOORSTELLING: PK (Vnr)

Vnr	Titel	Normaal	Abonnement
24	Ballet Don Quichote	40	35
25	Cowboy Billie Boem	15	12
26	Lebbis en Jansen	25	20
27	Aïda	50	45

FK: PROGRAMMA (Vnr) -> VOORSTELLING (Vnr)

4. Hieronder zie je de tabellen **Student** en **Studieresultaat** die onderdeel uitmaken van het studievolgsysteem van een ICT-opleiding. LET OP: Bij deze ICT-opleiding wordt elk vak afgesloten met precies één toets. Als een student een onvoldoende voor een toets, dan kan hij/zij die toets bij een volgende gelegenheid herkansen. De primary key van de tabel **Student** wordt gevormd door de kolom **studnr**.

Student

studnr	roepnaam	achternaam	startdatum	einddatum	vooropleiding
567422	Jip	Straatman	1-9-2016	31-8-2017	HAVO
567803	Britt	Dalen	1-9-2016	31-8-2017	VWO
570419	Rik	Janssen	1-9-2016		VMBO, MBO
586399	Mila	Verbeek	1-9-2017		HAVO
588197	Tom	Janssen	1-9-2017		HAVO, VWO
589477	Sophie	Ridder	1-9-2017		MBO

Studieresultaat

studnr	**vakcode**	vaknaam	**datum**	cijfer	dcode	dnaam
567422	Dtbs	Relationele Databases en SQL	5-12-2016	7,5	PTR	Potters
567803	Dtbs	Relationele Databases en SQL	5-12-2016	6,8	PTR	Potters
570419	Dtbs	Relationele Databases en SQL	5-12-2016	4,3	PTR	Potters
570419	Dtbs	Relationele Databases en SQL	12-3-2017	5,3	VRM	Vermeulen
586399	Dtbs	Relationele Databases en SQL	5-12-2016	5,1	PTR	Potters
586399	Dtbs	Relationele Databases en SQL	12-3-2017	6,4	VRM	Vermeulen
567422	Prgr1	Inleiding Programmeren	3-11-2017	6,9	KGM	Kaagman

studnr	**vakcode**	vaknaam	**datum**	cijfer	dcode	dnaam
567803	Prgr1	Inleiding Programmeren	3-11-2017	7,8	KGM	Kaagman
570419	Prgr1	Inleiding Programmeren	3-11-2017	5,6	KGM	Kaagman
586399	Prgr1	Inleiding Programmeren	3-11-2017	7,3	KGM	Kaagman
588197	Prgr1	Inleiding Programmeren	3-11-2017	2,9	VRM	Vermeulen
589477	Prgr1	Inleiding Programmeren	3-11-2017	8,4	VRM	Vermeulen

a. Staat de tabel **Student** in de 1e normaalvorm (1NV)? Motiveer je antwoord.

```
/*
Nee. De tabel staat in ONV omdat het niet-atomaire data bavat in de
kolom 'vooropleiding'.
*/
```

b. Zet de tabel **Student** indien nodig om naar tabellen in de 3e normaalvorm (3NV).

c. Breng voor de tabel Studieresultaat de functionele afhankelijkheden in kaart.

d. In welke normaalvorm staat de tabel Studieresultaat? Motiveer je antwoord.

e. Zet de tabel Studieresultaat indien nodig om naar tabellen in 3NV.

14 DIVERSE OEFENINGEN

14.1 MUZIEKDATABASE

1. Aan welke scholen zijn meer dan 2 componisten verbonden? Geef alle informatie over die scholen.

```
SELECT  *
FROM    Muziekschool
WHERE   schoolId IN (
    SELECT  schoolId
    FROM    componist
    GROUP   BY schoolid
    HAVING  COUNT(*) > 2);
```

2. Welke componisten hebben de meeste klassieke stukken geschreven? Geef id, naam en geboortedatum.

```
SELECT  componistId, naam, geboortedatum
FROM    Componist
WHERE   componistId IN (
    SELECT  componistId
    FROM    Stuk
    WHERE   genrenaam = 'klassiek'
    GROUP   BY componistId
    HAVING  COUNT(*) >= ALL (
        SELECT  COUNT(*)
        FROM    Stuk
        WHERE   genrenaam = 'klassiek'
        GROUP   BY componistId));

-- Of:
SELECT  componistId, naam, geboortedatum
FROM    Componist
WHERE   componistId IN (
    SELECT  componistId
    FROM    Stuk
    WHERE   genrenaam = 'klassiek'
    GROUP   BY componistId
    HAVING  COUNT(*) = (
        SELECT  TOP 1 COUNT(*)
        FROM    Stuk
        WHERE   genrenaam = 'klassiek'
        GROUP   BY componistId
        ORDER   BY 1 DESC));
```

3. Welke klassieke stukken hebben een onbekende bezetting? Geef stuknr en titel. Schrijf de query op twee manieren. Eén keer me een subquery, de andere met een join.

```
-- Subquery:
SELECT  stukNr, titel
FROM    Stuk
WHERE   genrenaam = 'klassiek'
        AND stuknr NOT IN (
            SELECT  stuknr
```

```
            FROM    Bezettingsregel);

-- Join:
SELECT  stukNr, titel
FROM    Stuk S
        LEFT JOIN Bezettingsregel B ON S.stukNr = B.stukNr
WHERE   genrenaam = 'klassiek'
        AND B.stukNr IS NULL;
```

4. Wie is de oudste componist?

```
SELECT  *
FROM    Componist
WHERE   geboortedatum = (
    SELECT  MIN(geboortedatum)
    FROM    Componist);
```

5. Wat is het jongste stuk?

```
SELECT  *
FROM    Stuk
WHERE   jaartal = (
    SELECT  MAX(jaartal)
    FROM    Stuk);
```

6. Je oma van 91 wil graag piano leren spelen. Gezien haar leeftijd wil ze graag een stuk voor beginners met een zo kort mogelijke speelduur leren spelen. Ze vraagt aan jou welk stuk het meest geschikt is. Ze heeft een brede muzieksmaak, dus genre maakt niet uit. Geef stuknr en titel.

```
-- Het bovenste deel gaat over het zoeken naar een stuk voor
-- beginners op de piano. De subquery gaat over het vinden
-- van een dergelijk stuk met de kortste speelduur.
SELECT  s.stuknr, s.titel
FROM    Stuk S
        INNER JOIN Bezettingsregel B ON S.stuknr = B.stuknr
WHERE   B.instrumentnaam = 'piano'
        AND S.niveaucode = 'A'
        AND speelduur <= (
            SELECT  MIN(s.speelduur)
            FROM    Stuk S
                    INNER JOIN Bezettingsregel B ON S.stuknr = B.stuknr
            WHERE   B.instrumentnaam = 'piano'
                    AND S.niveaucode = 'A');
```

14.2 OPENSCHOOL DATABASE

1. Welke docent begeleidt geen enkele cursus? Geef twee oplossingen: een met een subquery en een met een outer join.

```
-- Met subquery:
SELECT acr, naam
FROM   Docent
WHERE  acr NOT IN (
    SELECT  docent
    FROM    Begeleider);

-- En de outer join:
SELECT  acr, naam
FROM    Docent D
        LEFT JOIN Begeleider B ON D.acr = B.docent
WHERE  B.cursus IS NULL;
```

2. Welke docenten begeleiden 2 of meer cursussen? Geef twee oplossingen: een met een subquery en een met een join (maar zonder subquery).

```
SELECT  acr, naam
FROM    Docent
        INNER JOIN Begeleider ON acr = docent
GROUP   BY acr, naam
HAVING  COUNT(*) > 1    -- Of: COUNT(*) >= 2

SELECT  acr, naam
FROM    Docent
WHERE   acr IN (
    SELECT  docent
    FROM    Begeleider
    GROUP   BY Docent
    HAVING  COUNT(*) > 1);
```

3. Welke student heeft nog geen enkele cursus succesvol afgerond? Opmerking: Een cursus is succesvol afgerond als de student een voldoende heeft gehaald of een vrijstelling heeft gekregen.

```
SELECT  *
FROM    Student
WHERE   nr NOT IN (
    SELECT  student
    FROM    Inschrijving
    WHERE   cijfer >= 6 OR vrijstelling = 'J');
```

4. Geef de code en de omschrijving van cursussen waarvoor 2 of meer vrijstellingen zijn verleend. Geef twee SELECT-statements, een met een join en een met een subquery.

```
SELECT  code, naam
FROM    Cursus AS C
        INNER JOIN Inschrijving AS I ON C.code = I.cursus
WHERE   vrijstelling = 'J'
GROUP   BY code, naam
HAVING  COUNT(*) > 1);

SELECT  code, naam
FROM    Cursus
WHERE   code IN (
    SELECT  cursus
    FROM    Inschrijving
    WHERE   vrijstelling = 'J'
    GROUP   BY cursus
    HAVING  COUNT(*) > 1);
```

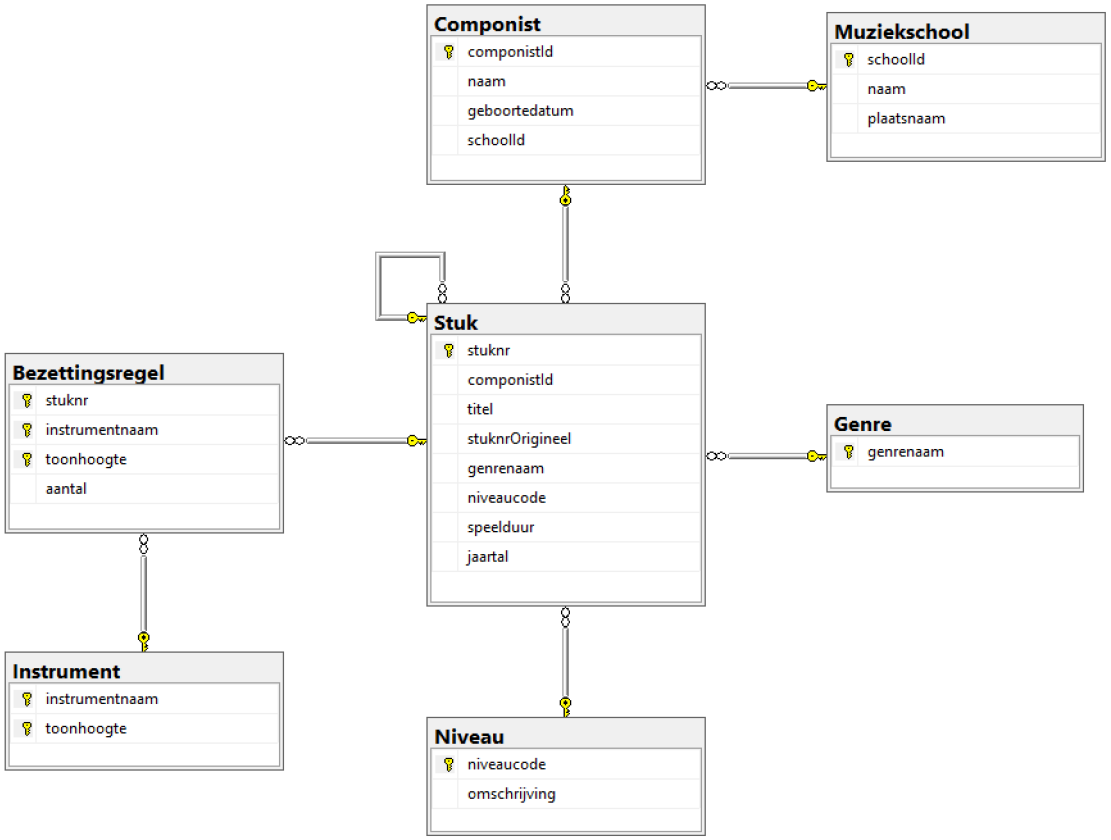
5. Geef voor elke student het aantal behaalde credits.

```
SELECT  S.nr, S.naam, SUM(C.credits)
FROM    Student S
        INNER JOIN Inschrijving I ON S.nr = I.student
        INNER JOIN Cursus C ON C.code = I.cursus
WHERE   cijfer >= 6
        OR vrijstelling = 'J'
GROUP   BY S.nr, S.naam;
```

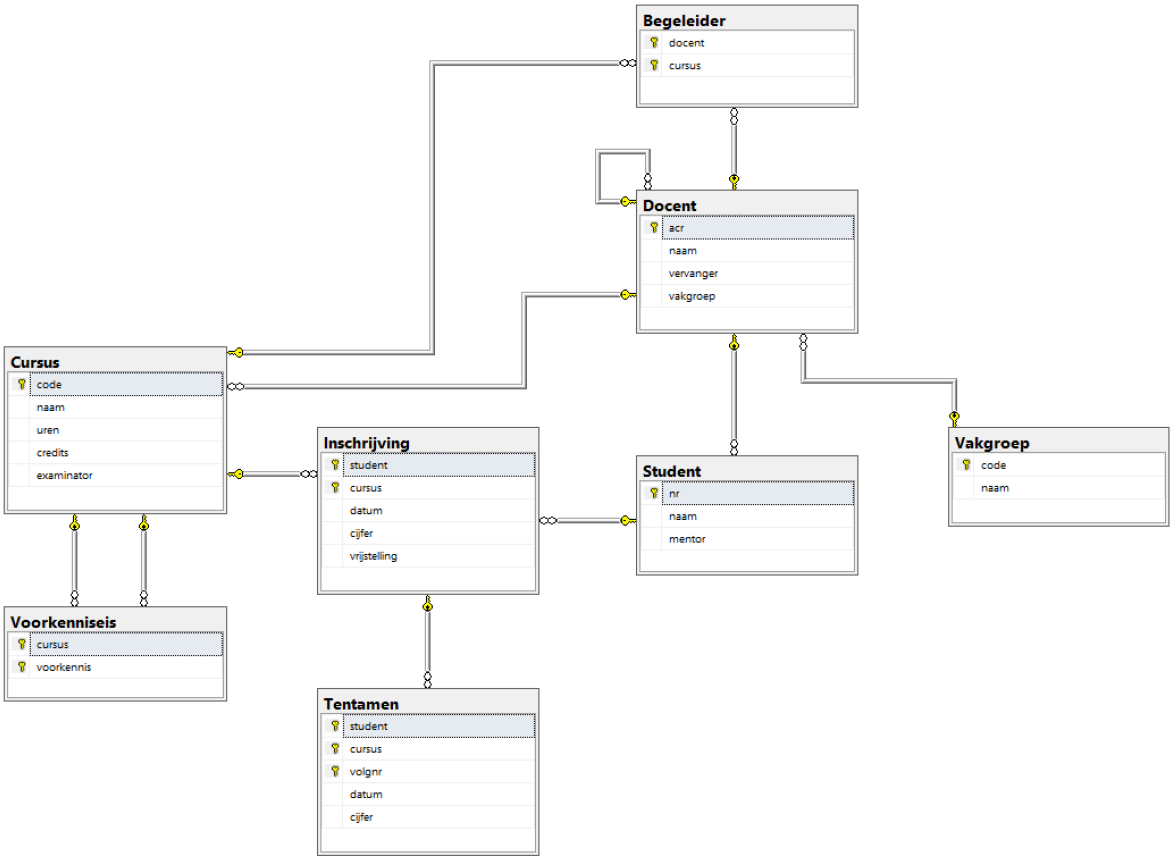
15 DATA MODELLEN

Hier vind je een uitleg van de databases die in het werkboek gebruikt worden. Per database vind je het schema en een korte beschrijving van de tabellen.

15.1 MUZIEKDATABASE



15.2 OPENSCHOOL



15.3 PREMIER DATABASE