

Databases

SQL

Niveau 4

Applicatieontwikkeling/ICT-beheerder

Inhoud

0	Inleiding	4
1	Database, datamodel en DBMS - wat is dat?	7
1.1	Inleiding	7
1.2	Database	7
1.3	Datamodel	8
1.4	Database Management Systeem	8
2	Structured Query Language.....	10
2.1	Inleiding	10
2.2	Geschiedenis van SQL	10
2.3	Databases en SQL.....	11
2.4	DML-statements	11
3	Het Relationele Model.....	13
3.1	Inleiding	13
3.2	Tabel, kolom en rij	13
3.3	Integriteitregel	13
3.4	Primaire sleutel.....	14
3.5	Kandidaatsleutel	14
3.6	Alternatieve sleutel.....	14
3.7	Refererende sleutel	15
3.8	Soorten relaties.....	15
4	De installatie van SQL Express 2012	18
4.1	Inleiding	18
4.2	Installatie SQL Express 2012	18
4.3	Configuratie van SQL Express	27
5	De oefendatabase	30
5.1	Inleiding	30
5.2	De database	30
5.3	Het maken van de oefendatabase.....	31
5.4	De database bekijken.....	33
5.5	Wat staat er in het script?	34
6	Het maken van een database	37
6.1	Inleiding	37
6.2	Syntax.....	37
6.3	Voorbeelddatabase.....	38
6.4	Het maken van een database	38
6.5	Het maken van een tabel.....	39
6.6	Het wissen van een database	41
6.7	Primaire sleutel.....	42
6.8	Het verwijderen van een primaire sleutel	44
6.9	De verwijzende sleutel.....	44
6.10	Samenvatting	47
7	Het selecteren van gegevens.....	49
7.1	Inleiding	49
7.2	Syntax.....	49

7.3	SELECT	49
7.4	De eenvoudigste versie van SELECT	50
7.5	Het bewaren van een of meer query's	53
7.6	Duplicaten vermijden.....	55
7.7	Commentaar gebruiken?	56
7.8	Selecteren met voorwaarden	57
7.9	Zoeken op meerdere voorwaarden	59
7.10	Een bijzonder geval.....	60
7.11	Wat als een veld leeg is?.....	61
8	De invoer van gegevens	63
8.1	Inleiding	63
8.2	Syntax.....	63
8.3	INSERT INTO	63
8.4	Invoer van gegevens in meerdere gerelateerde tabellen	66
9	Het wijzigen van gegevens.....	67
9.1	Syntax.....	67
9.2	Het statement UPDATE.....	67
9.3	Het aanpassen van een adres	68
9.4	Het wijzigen van gegevens uit meerdere gerelateerde tabellen	69
10	Het verwijderen van gegevens.....	71
10.1	Syntax.....	71
10.2	Het statement DELETE	71
10.3	TRUNCATE.....	73
10.4	Samenvatting	73
11	Het selecteren van gegevens deel II.....	74
11.1	Inleiding	74
11.2	Syntax.....	74
11.3	De uitvoer sorteren.....	74
11.4	De uitvoer groeperen.....	76
11.5	De groepering beperken	78
11.6	Andere ingebouwde functies.....	79
12	Meer dan een tabel gebruiken en subquery's	81
12.1	Inleiding	81
12.2	Syntax.....	81
12.3	Gegevens uit meer dan een tabel.....	82
12.4	De JOIN-query.....	82
12.5	Het Cartesiaanse product	83
12.6	De subquery	83
12.7	De IN-operator	84
12.8	Het pseudoniem.....	85

0 Inleiding

In deze lessen leer je werken met databases en SQL-code. Dit lesmateriaal is zowel geschikt voor de opleiding Applicatieontwikkelaar niveau 4, als voor de opleidingen ICT-Beheerder en Netwerkbeheerder. In de studiewijzer (bijlage A) is weergegeven welke onderdelen voor welke opleiding van belang zijn. Natuurlijk kan ook alles doorgenomen en bestudeerd worden.

In dit lesmateriaal staat beschreven hoe databases in SQL-code opgebouwd kunnen worden. Met behulp van SQL zijn gegevens uit databases te halen, te muteren, te wissen en in te voegen. Ook het beheer van databases, zoals back-up/restore, maintenance plannen, autorisatie en koppelingen met andere software, komt aan bod. Vanwege de hoeveelheid onderwerpen is dit lesmateriaal verdeeld in drie onderdelen.

- 1 Basis databases en SQL
- 2 Complexere oplossingen
- 3 Beheren van een databaseserver

Bij de delen *Basis SQL* en *Complexere oplossingen* wordt Microsoft SQL Server Express 2012 gebruikt om het een ander uit te voeren. In het deel *Beheren* wordt Microsoft SQL Server 2012 toegepast. Deze server is het best te installeren in een virtuele omgeving. Microsoft Word, Excel en Access (versie 2010 of hoger) dienen dan ook beschikbaar te zijn in deze virtuele omgeving.

Het lesmateriaal is opgebouwd uit een aantal hoofdstukken. Bij elk hoofdstuk behoren een of meer opdrachten. Deze opdrachten moeten gemaakt worden. De uitwerking van deze opdrachten wordt besproken in de les.

LET OP!

In dit lesmateriaal wordt uitgegaan van **ANSI-SQL**. Dat betekent dat de gebruikte statements in elke programmeertaal of databaseomgeving uit te voeren zouden moeten zijn.

In het lesmateriaal staan regelmatig teksten die ingevoerd moeten worden op de computer. Deze teksten zijn in *cursief* lettertype weergegeven, zodat duidelijk is wat nu wel of juist niet moet worden getypt.

Gewone tekst
Tekst die ingevoerd moet worden.

Tekstboxen, zoals hierboven, worden gebruikt om belangrijke punten nog eens extra onder de aandacht te brengen. Deze opmerkingen/statements moet je beheersen om verder te kunnen gaan met de lesstof.

In het lesmateriaal wordt het muissymbool  gebruikt om aan te geven dat er een handeling verricht moet worden.

Alle hoofdstukken in dit lesmateriaal hebben dezelfde opbouw.

- Na de inleiding krijg je te zien wat van je wordt verwacht als je het hoofdstuk hebt doorgewerkt.
- In het tweede onderdeel krijg je de syntax (= schrijfwijze) te zien van alle in dit hoofdstuk besproken commando's/statements.
- Vervolgens komt de lesstof aan bod.

Deel I

Databases en SQL

De basis

1 Database, datamodel en DBMS - wat is dat?

1.1 Inleiding

In dit hoofdstuk wordt kort omschreven wat een database is en waarom het belangrijk is dat applicatieontwikkelaars (AO) en beheerders hiermee om kunnen gaan. De uitleg is voor een belangrijk deel een herhaling van de lesstof die je eerder hebt gekregen bij Basisinformatica/ECDL (Access).

Na bestudering van dit hoofdstuk moet je tot het volgende in staat zijn.

- Het kunnen beschrijven wat een DBMS is.
- Je kunt uitleggen wat een database is, waaruit deze bestaat en waar je een database voor kunt gebruiken.
- Het kunnen lezen van een Bachmandiagram.
- De begrippen integriteit en redundantie kunnen gebruiken en kunnen toelichten in relatie tot databases.

1.2 Database

Het woord database wordt voor verschillende begrippen gebruikt.

- Opgeslagen gegevens (= database).
- De manier waarop gegevens zijn opgeslagen (= datamodel).
- De software om databases te maken en te benaderen (= Database Management Systeem of DBMS).

In dit lesmateriaal betekent 'database' echter altijd 'de opgeslagen gegevens'. Maar wat is een database dan?

Een database is een archief: digitaal opgeslagen en ingericht om flexibel te raadplegen en te gebruiken.

Databases spelen een belangrijke rol bij het archiveren en actueel houden van gegevens bij onder meer de overheid, financiële instellingen en bedrijven, in de wetenschap en worden daarnaast op kleinere schaal ook privé gebruikt.

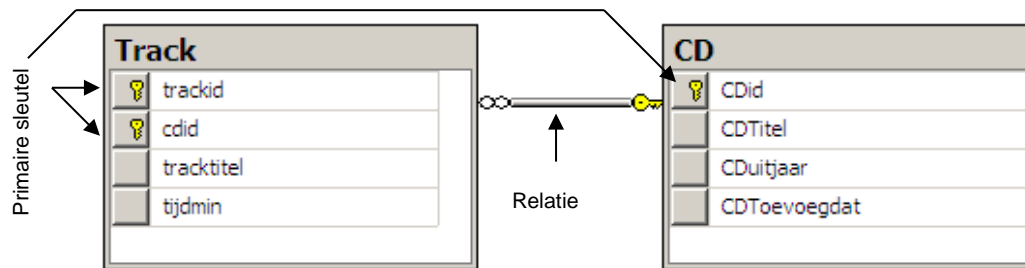
Een database is pas een database als het aan de volgende minimale voorwaarden voldoet.

- Gegevens moeten eenvoudig kunnen worden opgeslagen.
- Gegevens moeten eenvoudig kunnen worden opgezocht en doorzocht.
- Gegevens moeten gewijzigd kunnen worden.
- Gegevens moeten verwijderd kunnen worden zonder dat dat de werking van het systeem nadelig beïnvloedt.

1.3 Datamodel

Met een datamodel (of gegevensmodel of Bachmandiagram) wordt beschreven hoe de gegevens in een informatiesysteem gestructureerd zijn. In een datamodel kun je zien welke gegevens in een informatiesysteem worden vastgelegd en wat de verbanden zijn tussen deze gegevens.

Hieronder staat een datamodel van een cd-database.



Een datamodel bestaat dus uit:

- tabellen;
- velden;
- sleutels;
- de relaties tussen de tabellen.

1.4 Database Management Systeem

Een DBMS zal nooit uit zichzelf de gegevens in een database wijzigen of verwijderen. Iemand of iets zal het DBMS daartoe opdracht moeten geven. Dit kan jij zijn, als gebruiker, maar het kan ook een programma zijn. In het tweede geval is het programma de gebruiker.

Met behulp van speciale talen worden opdrachten aan een DBMS gegeven. Dit soort talen wordt databasetalen genoemd. Opdrachten worden door gebruikers ingevoerd en door het DBMS verwerkt. Elk DBMS, van welke fabrikant dan ook, bezit een databasetaal. Sommige systemen hebben zelfs meer dan een databasetaal. Tussen al die talen bestaan verschillen. Deze verschillende talen zijn te verdelen in groepen. Een van deze groepen wordt gevormd door de relationele databasetalen.

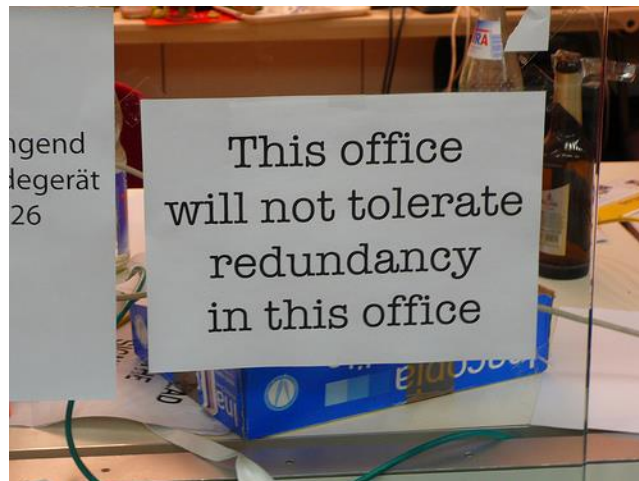
De belangrijkste relationele databasetaal is SQL (Structured Query Language). Deze taal wordt door vrijwel alle DBMS'en ondersteund. Naast de standaard-SQL breiden DBMS-fabrikanten de taal nog uit met eigen specifieke opdrachten. Deze opdrachten zijn meestal niet uitwisselbaar met andere databaseomgevingen.

Een belangrijke taak van een DBMS is het handhaven van de integriteit van de databasegegevens. Hiermee wordt bedoeld dat het DBMS ervoor moet zorgen dat databasegegevens altijd voldoen aan regels die in de werkelijkheid gelden.

Als bijvoorbeeld een werknemer maar voor een afdeling tegelijk mag werken, mag nergens in de database geregistreerd kunnen worden dat deze werknemer voor twee of meer afdelingen werkt.

Met integriteit wordt ook bedoeld dat twee verschillende databasegegevens elkaar niet tegenspreken (dit wordt ook wel consistentie van gegevens genoemd).

Tevens moet het niet mogelijk zijn dezelfde gegevens meermalen in te voeren. Dit noemen we redundantie.



Daarnaast mogen twee verschillende databasegegevens elkaar niet tegenspreken. Dit wordt ook wel inconsistentie van gegevens genoemd.

2 Structured Query Language

2.1 Inleiding

In dit hoofdstuk wordt kort ingegaan op wat SQL is en waar het vandaan komt. Daarnaast komen de drie verschillende typen SQL aan bod.

Na bestudering van dit hoofdstuk moet je tot het volgende in staat zijn.

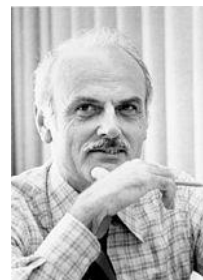
- Het kunnen omschrijven wat SQL is en waarvoor het gebruikt kan worden.
- Het kunnen omschrijven wat ANSI is en wat dit te maken heeft met SQL.
- De drie soorten SQL kunnen omschrijven en kunnen aangeven wat voor soort opdrachten bij welke soort horen.

2.2 Geschiedenis van SQL

De geschiedenis van SQL begint als de Britse computerwetenschapper Dr. Edgar F. Codd in juni 1970 een artikel publiceert met als naam 'A Relational Model of Data for Large Shared Data Banks'. De inhoud van dit artikel wordt vervolgens geaccepteerd als het uiteindelijke model voor relationele datasystemen.

Aan de hand van dit relationele model bouwt IBM een experimenteel systeem, genaamd 'System R'.

Donald Chamberlin en Raymond Boyce ontwikkelen de taal 'Structured English Query Language', of SEQUEL, omdat er ook een mogelijkheid moet zijn om data in 'System R' te lezen en te bewerken. Helaas zat er aan het woord SEQUEL een geregistreerd handelsmerk van een Engelse vliegtuigbouwer en daarom is, om de concepten van de taal te kunnen publiceren, de naam veranderd naar SQL.



Edgar F. Codd
(1923 - 2003)

Hoewel er na publicatie van het artikel van Codd een aantal relationele databases verschijnt, maakt IBM zich in 1978 op om het eerste relationele databasesysteem met SQL als onderdeel van System/38 op de markt te zetten. Als dit in 1979 daadwerkelijk gebeurt, blijkt IBM niet meer de eerste te zijn. Relational Software is IBM een paar weken te vlug af met de introductie van Oracle. Enkele jaren later neemt Relational Software de naam Oracle over als bedrijfsnaam.

De standaard-SQL is er sinds 1986, In dat jaar brengt het ANSI (American National Standards Institute) de eerste versie van de SQL-standaard uit en in 1987 wordt deze standaard ook erkend door het ISO. Na een kleine uitbreiding van de standaard in 1989 volgen er in 1992 en 1999 twee forse herzieningen. In dit lesmateriaal wordt de ANSI 1999 standaard gebruikt. Voor meer informatie zie: <https://nl.wikipedia.org/wiki/SQL>

2.3 Databases en SQL

Wat gaat er in een database? In ieder geval niet zomaar wat data. Deze data, of gegevens hebben structuur, eigenschappen en mogelijk ook relaties met andere gegevens. Die gegevens hebben ook weer structuur en eigenschappen. De termen data en gegevens zijn uitwisselbaar.

SQL-opdrachten, ook wel statements genoemd, kunnen op twee manieren worden gebruikt. Enerzijds kunnen ze door middel van een programma worden ingetypt en direct worden uitgevoerd, anderzijds kunnen ze worden opgenomen in een derde, vierde of vijfde-generatietaal (3GL, 4GL, 5GL) en direct worden uitgevoerd. In dit laatste geval spreken we van 'embedded SQL', of 'ingebod' SQL.

SQL-opdrachten zijn te onderscheiden naar drie typen.

- Data Definition Language (DDL)
- Data Control Language (DCL)
- Data Manipulation Language (DML)

Als er nog geen database beschikbaar is, moet deze eerst worden aangemaakt. Hiertoe kunnen DDL-statements worden gebruikt. Anders gezegd: met behulp van DDL-statements kan de structuur van de database worden beheerd. Dit is dan ook het eerste onderdeel dat wordt behandeld in dit lesmateriaal.

Is een database eenmaal beschikbaar, dan moet je kunnen bepalen wie er met de ingevoerde gegevens moet kunnen werken. Er moeten rechten kunnen worden toegekend aan gebruikers. Het beheer van de rechten wordt geregeld met behulp van DCL-statements.

DML-statements worden over het algemeen door gebruikers toegepast. Zij selecteren gegevens op basis van de bestaande informatiebehoefte, voeren nieuwe gegevens in of wijzigen en verwijderen gegevens.

De eerste twee typen statements (DDL en DCL) worden voornamelijk gebruikt door de database-administrator (DBA) en de applicatieontwikkelaar. De DBA beheert de inhoud van de database en de toegang tot de database.

2.4 DML-statements

Databases worden gebruikt voor het opslaan van gegevens (data). Het is de bedoeling dat alle ingevoerde gegevens kunnen worden gebruikt. Dat is immers de reden geweest om deze gegevens in te voeren. De gegevens hebben geen of niet altijd betekenis. Met behulp van DML-statements kan bijvoorbeeld uit de gegevens een selectie worden gemaakt, waarna de nieuwe set gegevens (de geselecteerde gegevens) wel betekenis voor de gebruiker kan hebben.

Naast het statement voor het selecteren van gegevens zijn er DML-statements voor het invoeren, bewerken, bijwerken en verwijderen van gegevens.

Hieronder zijn alle elementen (volgens ANSI 99) weergegeven, die gebruikt kunnen worden bij het SELECT-statement. Met dit statement kun je gegevens selecteren of laten zien binnen een database.

<ul style="list-style-type: none"> • ALL • AND • ANY • AS • AVG() • BETWEEN • CNCAT() • CUNT() • CRSS JIN • CURRENT_DATE() • CURRENT_TIME() • CURRENT_TIMESTAMP() • DATE() • EXISTS • FULL UTER JIN • GETDATE() • GRUP BY • HAVING • IN • IN • INNER JIN • INSERT • INSTR() • LCASE() • LEFT UTER JIN 	<ul style="list-style-type: none"> • LEFT() • LEN() • LIKE • LIMIT • MAX() • MIN() • NT • NW() • NULL • R • RDER BY • REPLACE • REPLACE() • RIGHT UTER JIN • RIGHT() • RUND() • SELECT • SELECT DISTINCT • SELF JIN • SUM() • TP • UCASE() • UPDATE • WHERE • YEAR()
--	---

Vaak worden SQL-statements weergegeven in zogenoemde syntaxdiagrammen. In dergelijke diagrammen kun je de opbouw lezen van een statement. Deze diagrammen zijn vaak terug te vinden in de helpteksten van de leverancier.

3 Het Relationele Model

3.1 Inleiding

In dit hoofdstuk wordt dieper ingegaan op wat een database is en waaruit deze is opgebouwd. Daarnaast komen de termen sleutel en relatie aan bod.

Na bestudering van dit hoofdstuk moet je tot het volgende in staat zijn.

- Het kunnen omschrijven uit welke onderdelen een database is opgebouwd door de termen tabel, record en veld te gebruiken.
- Het kunnen aangeven wat een primaire sleutel is en waarvoor deze gebruikt wordt.
- Het verband kunnen uitleggen tussen een foreign key en relaties.
- Weten welke andere typen sleutels er nog meer zijn en waarvoor deze dienen.
- Het kunnen aangeven van de verschillende relatiesoorten en wanneer deze toe te passen.

3.2 Tabel, kolom en rij

Gegevens in een relationele database kun je maar op een manier bewaren: je moet alles opslaan in tabellen. De officiële naam voor een tabel is eigenlijk relatie. Van deze naam stamt ook de term relationeel model af. Er is echter voor de term tabel gekozen, omdat deze door SQL gebruikt wordt.

Hieronder staat een voorbeeld van de cd-tabel. Deze tabel bevat gegevens over drie cd's.

IXP-044457\SQ...SS.CD - dbo.CD		IXP-044457\S... - Diagram_0*			Sur
	CDId	CDTitel	CDuitjaar	CDToevoegdat	
	1	Hitzone 44	2008	13-3-2008 9:15:48	
	2	Hitzone 45	2008	31-5-2008 8:43:56	
	3	Hitzone 50	2009	21-6-2009 8:44:15	
►*	NULL	NULL	NULL	NULL	

CDId, *CDTitel*, *CDuitjaar* en *CDToevoegdat* zijn de namen van de kolommen in de tabel. De *CDId*-kolom bevat de waarden 1, 2 en 3. Dit is een verzameling waarden en wordt ook wel de populatie van de *CDId*-kolom genoemd. De cd-tabel bevat drie rijen, één voor elke cd.

Een tabel heeft twee speciale eigenschappen.

- De kruising van een rij en een kolom kan maar één waarde bevatten.
- De rijen in een tabel hebben geen specifieke volgorde. Je kunt dus nooit spreken over de eerste rij, de laatste drie of de volgende rij. De inhoud van een tabel is een verzameling rijen. De volgorde hiervan bepaal je met SQL.

3.3 Integriteitregel

De inhoud van een tabel behoort meestal aan bepaalde regels te voldoen, de zogenaamde integriteitregels. Twee voorbeelden van integriteitregels: het spelersnummer van een

voetballer mag niet negatief zijn, en twee voetballers mogen niet hetzelfde spelersnummer hebben.

Integriteitregels worden door een relationeel databasemanagementsysteem (RDBMS) gecontroleerd. Elke keer dat de inhoud van een tabel gemuteerd wordt, controleert het RDBMS of de nieuwe gegevens aan de geldende integriteitregels voldoen. Dit is duidelijk een taak van het RDBMS. De integriteitregels moeten dan wel eerst gespecificeerd worden, zodat ze bekend zijn.

Integriteitregels kunnen allerlei vormen hebben. Een aantal komt in de praktijk zo vaak voor dat ze speciale namen hebben gekregen.

- Primaire sleutel
- Kandidaatsleutel
- Alternatieve sleutel
- Refererende sleutel

3.4 Primaire sleutel

De primaire sleutel van een tabel is een kolom in een tabel (of een combinatie van een aantal kolommen) die gebruikt kan worden als unieke identificatie voor de rijen in die tabel.

Twee verschillende rijen in een tabel mogen dus nooit in de primaire sleutel dezelfde waarde hebben en in elke rij moet de primaire sleutel altijd één waarde hebben. De *CDid*-kolom in de eerder getoonde cd-tabel kun je beschouwen als de primaire sleutel van die tabel. Twee cd's mogen dus nooit hetzelfde nummer hebben en er zal nooit een cd mogen zijn zonder *CDid*.

3.5 Kandidaatsleutel

Sommige tabellen bezitten meer dan een kolom (of combinatie van kolommen) die aangewezen zou kunnen worden als primaire sleutel. Zij bezitten allemaal de eigenschappen van een primaire sleutel. Deze kolommen (of combinatie van kolommen) worden kandidaatsleutels genoemd. Slechts een daarvan wordt echter aangewezen als de primaire sleutel. Een tabel heeft dus altijd minimaal één kandidaatsleutel.

Uitgaande van het feit dat in de cd-tabel de *CDToevoegdat* van elke cd uniek is, maakt dat dit veld ook een kandidaatsleutel is. Dit veld is dus ook aan te wijzen als primaire sleutel.

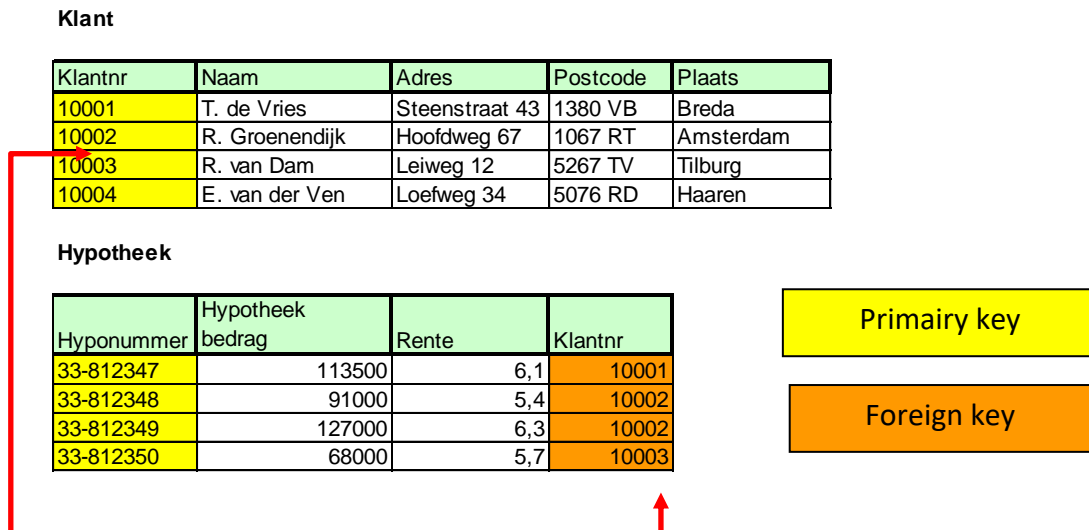
3.6 Alternatieve sleutel

Een kandidaatsleutel die niet de primaire sleutel van een tabel vormt, wordt een alternatieve sleutel genoemd. Het begrip kandidaatsleutel is dus een verzamelnaam voor alle primaire en alternatieve sleutels. De kolom *CDToevoegdat* uit het vorige voorbeeld is een alternatieve sleutel.

3.7 Refererende sleutel

Een refererende sleutel is een kolom (of combinatie van kolommen) in een tabel waarvan de populatie een deelverzameling is van de populatie van de primaire sleutel van een tabel. Andere benamingen voor refererende sleutel zijn: verwijzende sleutel, vreemde sleutel, foreign key of referential key.

Waarschijnlijk weet je nu nog steeds niet wat een foreign key is. Hieronder is een voorbeeld weergegeven.



De tabel *Hypotheek* heeft een relatie met de tabel *Klant*. Het veld *Klantnr* uit de tabel *Hypotheek* verwijst naar het veld *Klantnr* uit het tabel *Klant*.

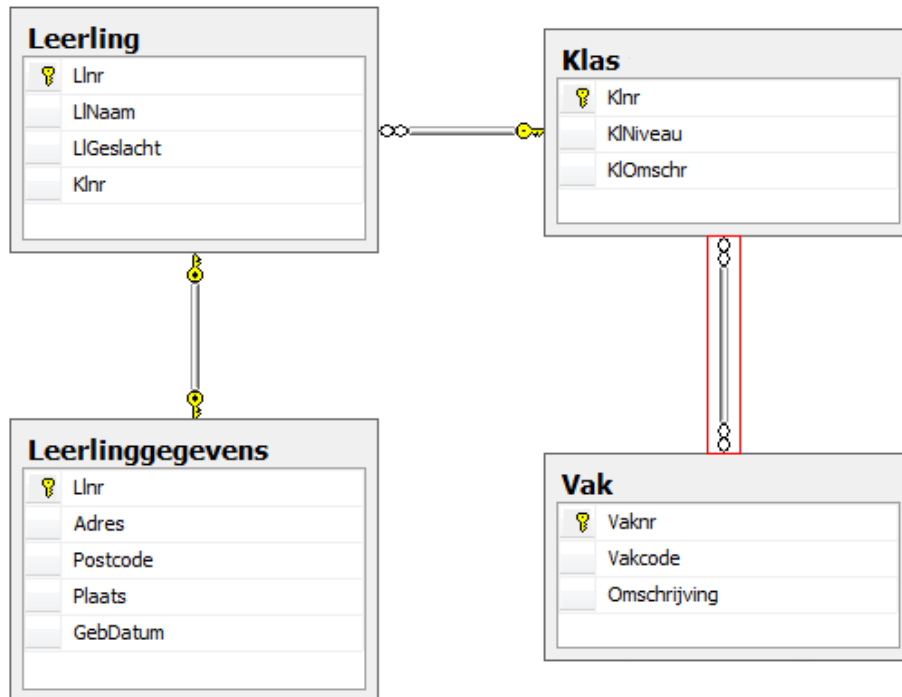
Het veld *Klantnr* in de tabel *Hypotheek* wordt een refererende sleutel genoemd.

3.8 Soorten relaties

Binnen een relationele database heeft elke tabel een relatie met een of meer andere tabellen. Er zijn vier verschillende relaties die je kunt tegenkomen.

- 1 op 1
- 1 op n (= 1 op veel)
- n op 1 (= veel op 1)
- n op m (= veel op veel)

Het volgende plaatje toont een stukje van een databasediagram waarin de verschillende relatiesoorten voorkomen.



De ‘veel-op-veel’-relatie (= rood) is wel getekend, maar kan standaard niet worden aangemaakt in SQL.

De relatie die je het meest tegenkomt binnen databases, is de ‘1-op-veel’- of ‘veel-op-1’-relatie. In het bovenstaande databasediagram ligt er tussen de tabellen *Leerling* en *Klas* een dergelijke relatie. Een klas kan diverse leerlingen bevatten.

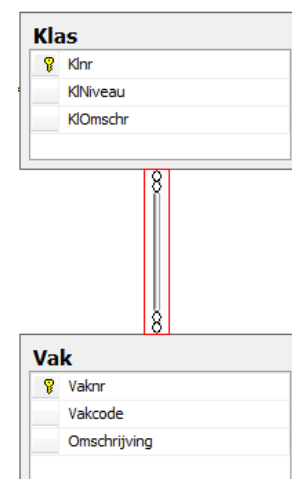
Een ‘1-op-1’-relatie kom je zelden tegen in een database. Het kan wel, maar vaak is dit een teken dat het databaseontwerp nog verbeterd dient te worden. In het voorbeeld ligt deze relatie tussen de tabellen *Leerling* en *Leerlinggegevens*. Een leerling heeft een adres, postcode, geboortedatum en dergelijke.

Een ‘veel-op-veel’-relatie kan wel uit een ontwerp komen, maar kan in de praktijk NOOIT in een database worden gebouwd. Hiervoor moet gebruik worden gemaakt van een zogenaamde koppeltabel. In deze koppeltabel zitten de primaire sleutelvelden van beide tabellen die worden gekoppeld. Deze velden worden de primaire sleutel van de koppeltabel.

Hiernaast zie je de tabellen *Klas* en *Vak*.

Een klas kan een vak volgen, maar een vak kan ook door verschillende klassen gevolgd worden. Hier is dus sprake van een ‘veel-op-veel’-relatie. Dit mag en kan niet in een database.

Een koppeltabel moet daarom tussen beide tabellen geplaatst worden.





In de koppeltabel zijn ook nog twee velden toegevoegd, waardoor er een rooster kan worden gegenereerd.

Maak opdracht 3.1 + 3.2.

4 De installatie van SQL Express 2012

4.1 Inleiding

Je hebt nu de theoretische kant van het verhaal gelezen. In dit hoofdstuk wordt kort toegelicht hoe je SQL Express 2012 dient te installeren. Deze versie gebruik je om alle volgende opdrachten te maken. Je gebruikt versie 2012, omdat met deze versie alle opdrachten gemaakt kunnen worden. Natuurlijk is het altijd mogelijk om een hogere versie te gebruiken. Bedenk dan wel dat de verschillende screenshots er anders uit kunnen zien.

Na bestudering van dit hoofdstuk moet je tot het volgende in staat zijn.

- Het kunnen installeren en configureren van SQL Express 2012.
- Het verschil kunnen toelichten van mix-mode en Windows Authentication.

4.2 Installatie SQL Express 2012

De installatiebestanden zijn te downloaden via het Dreamsparkaccount. Zorg er wel voor dat je de Express-editie with advanced services downloadt. Als je een andere versie downloadt, kan het zijn dat je de grafische beheerssoftware mist.

Kies de juiste versie, 32- of 64-bits, op basis van jouw operating system. Het is bekend dat bij een aantal systemen de installatie van de 64-bit-versie niet lukt. Installeer dan gewoon de 32-bit-versie.

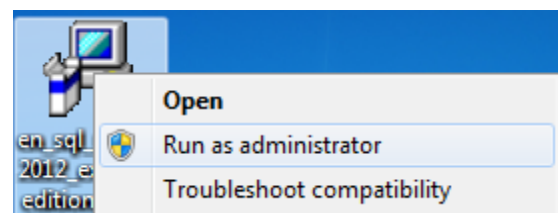
Systeemeisen

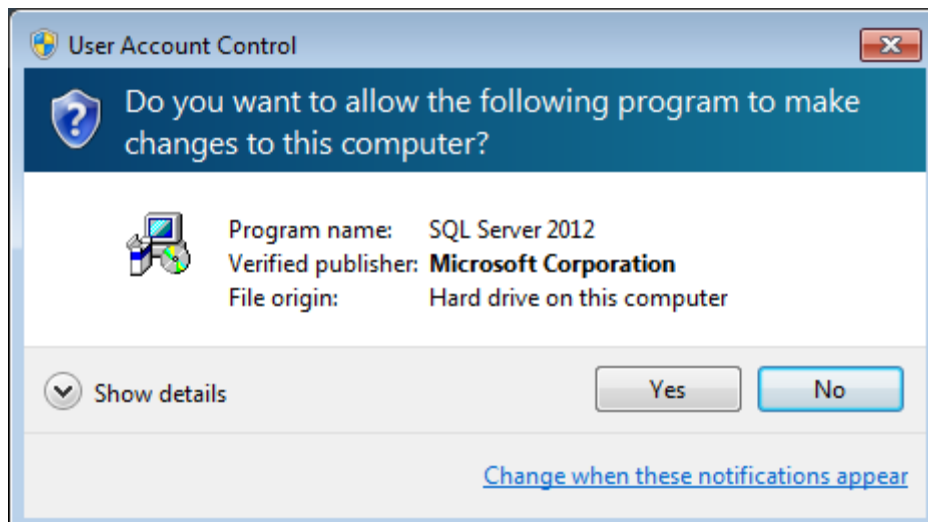
- CPU 1 GHz of hoger (32-bit)
- CPU 1.4 GHz of hoger (64-bit)
- Minimaal 512 MB RAM (2 GB is aanbevolen)
- 2,2 GB ruimte op de harddisk

Mocht je al een versie van SQL op je systeem hebben staan, dan is het raadzaam deze te verwijderen voor je met de onderstaande installatie verder gaat.

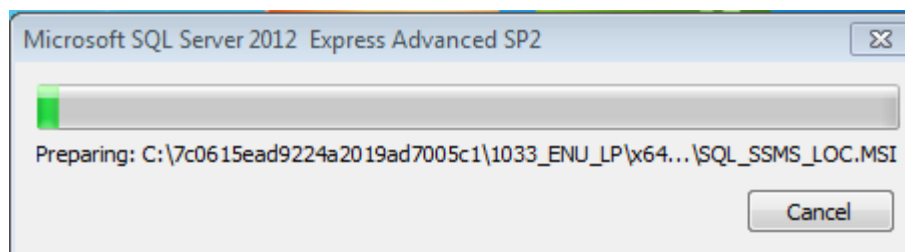
☞ Selecteer de juiste versie (32-/64-bits) en voer het installatiebestand als administrator uit. Klik daarvoor met de rechtermuisknop op het bestand.

☞ Bevestig de installatie met de optie **Yes**.



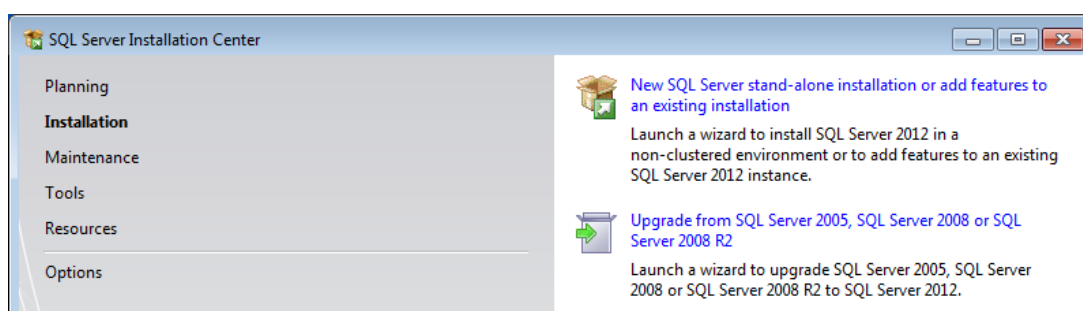


Er wordt nu een aantal bestanden klaargezet, waarna met de werkelijke installatie kan worden begonnen.



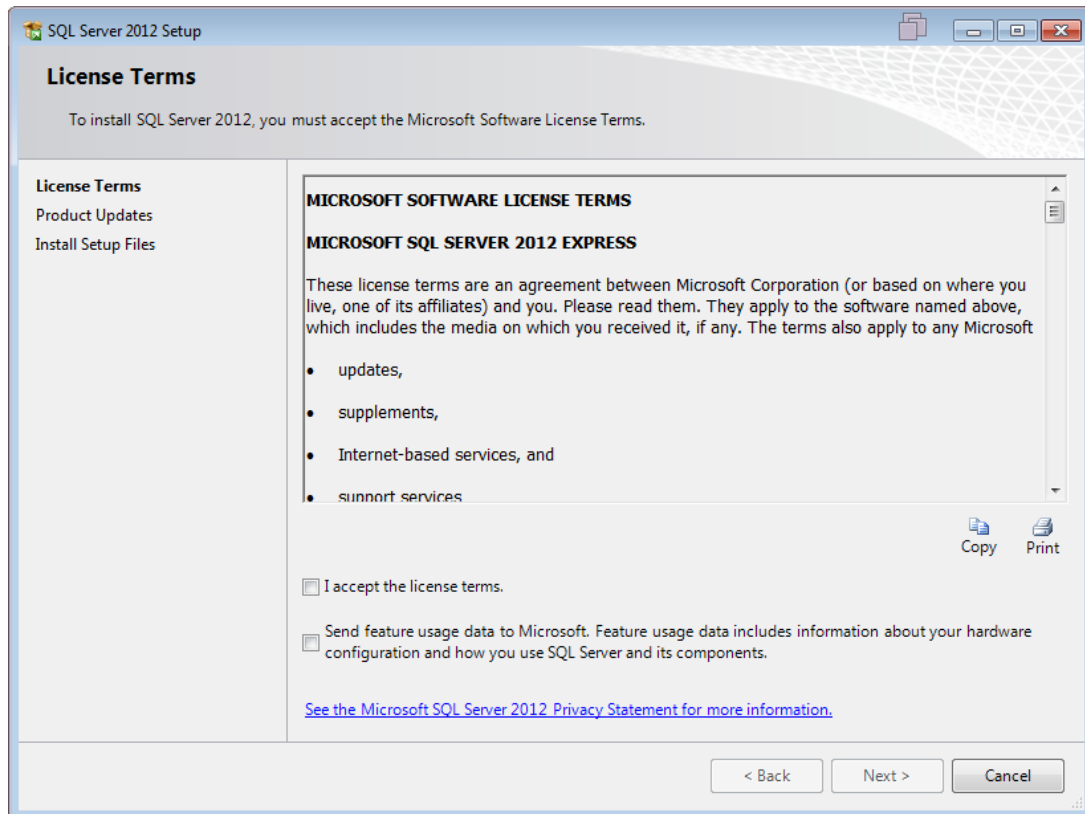
Na enige tijd verschijnt het SQL Server Installation Center.

Mocht er al een oudere versie van SQL zijn geïnstalleerd, dan dient de optie **Upgrade from Server 2008 R2** gekozen te worden. In dit lesmateriaal wordt de betreffende optie niet nader uitgelegd.



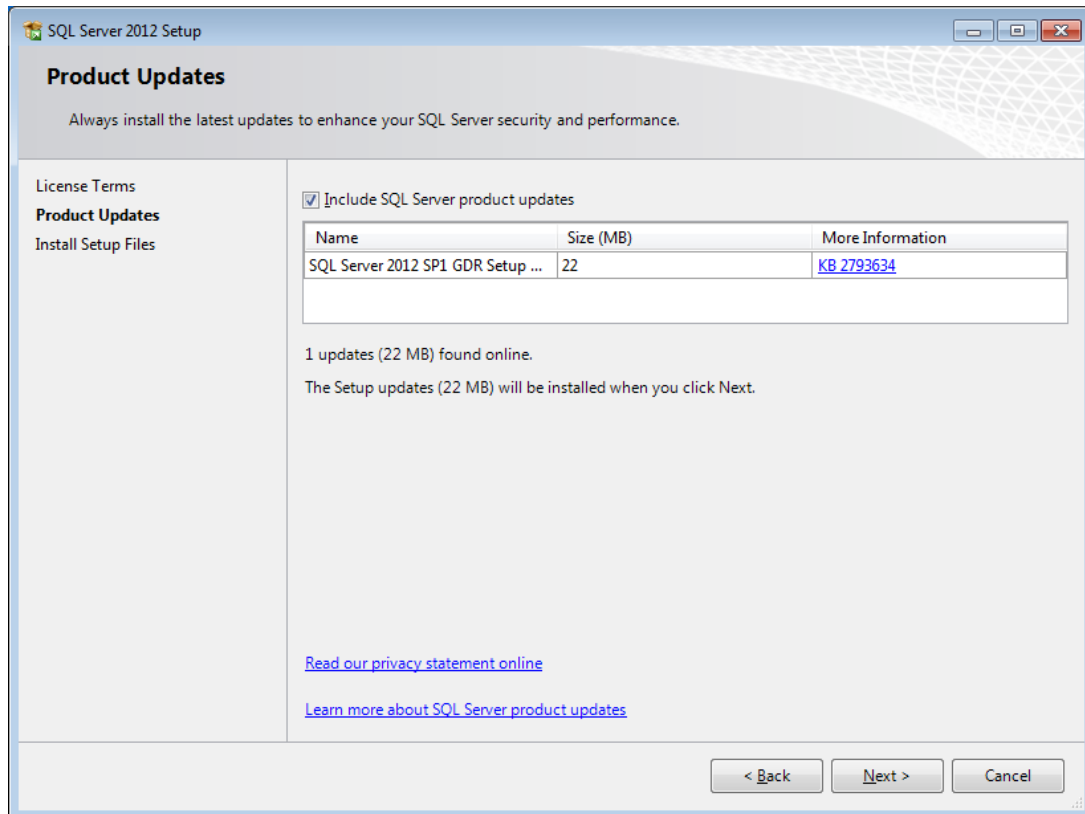
☞ Kies de optie **New SQL Server ... installation**.

Het scherm met de licentievoorwaarden verschijnt.



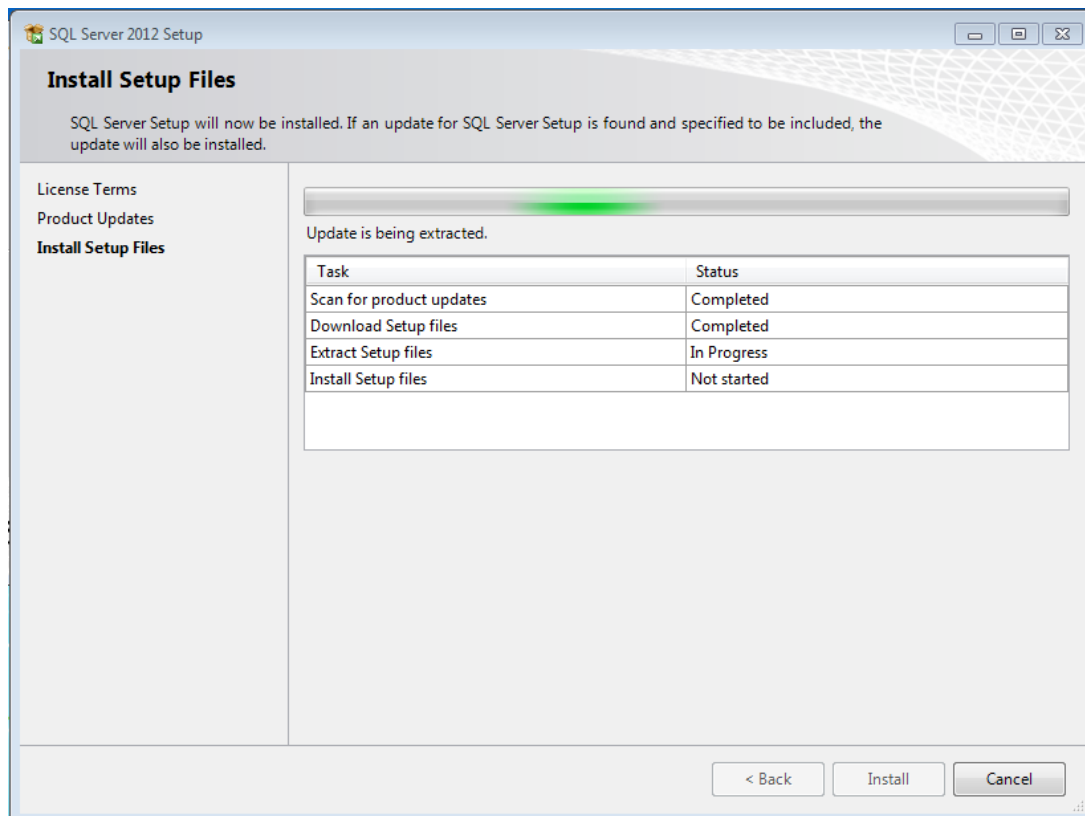
☞ Vink de optie ***I accept the license terms*** aan en druk op de knop ***Next*** om te verder te gaan.

Als je een internetverbinding hebt, dan kijkt de installer of er nog belangrijke updates zijn. Als deze er zijn, dan zullen deze automatisch worden weergegeven.

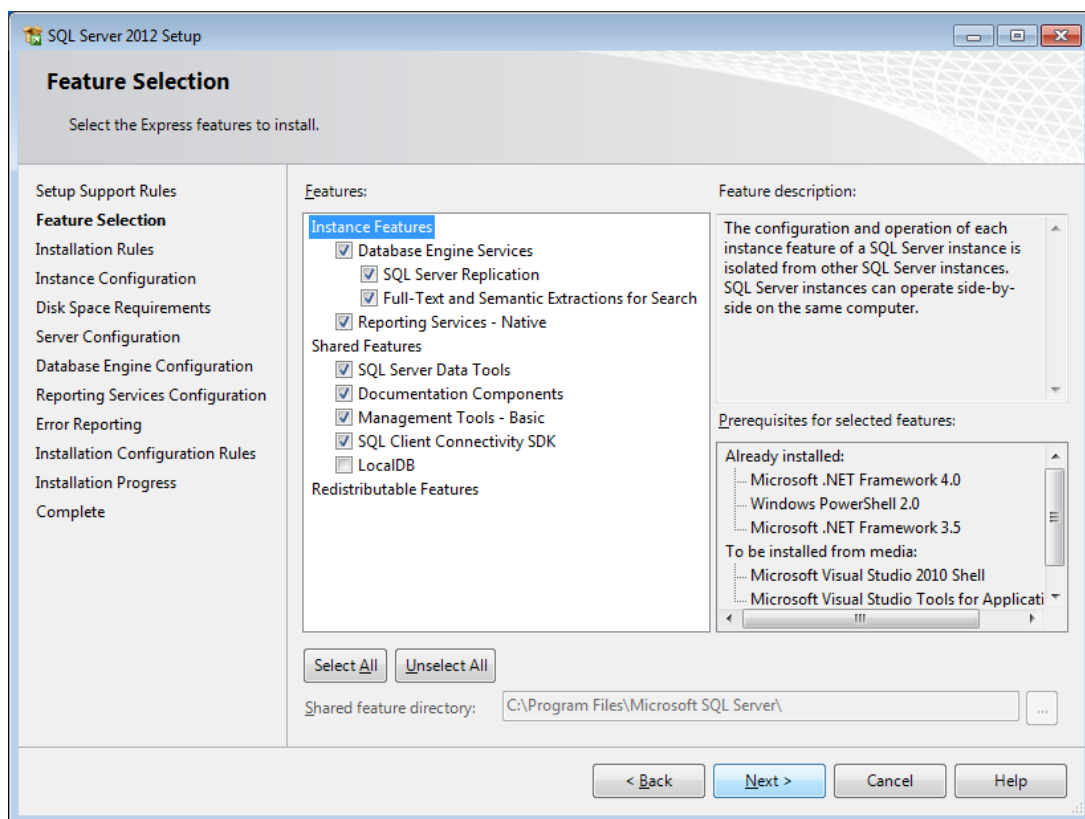


☞ Druk op **Next** om verder te gaan.

De update wordt nu gedownload en geïnstalleerd. Als dit gereed is, gaat de installatie van SQL Express verder. Je kunt de melding krijgen dat de computer na installatie opnieuw moet worden opgestart.



Nu moet je de onderdelen aangeven die geïnstalleerd moeten worden. In de onderstaande schermafdruck zie je, dat de optie **LocalDB** niet is geselecteerd. Deze optie installeert een zogenoemde lightversie van SQL Express. Die optie zal niet worden gebruikt.



☞ Selecteer alle opties om te installeren, behalve *LocalDB* en druk op **Next**.

Je moet nu de naam van de server opgeven. Standaard is dit *SQLExpress*. Neem de instellingen over zoals deze in de onderstaande afbeelding zijn weergegeven en druk op **Next**.

SQL Server 2012 Setup

Instance Configuration

Specify the name and instance ID for the instance of SQL Server. Instance ID becomes part of the installation path.

Setup Support Rules
Feature Selection
Installation Rules
Instance Configuration
Disk Space Requirements
Server Configuration
Database Engine Configuration
Reporting Services Configuration
Error Reporting
Installation Configuration Rules
Installation Progress
Complete

☐ Default instance
☒ Named instance:

Instance ID:

Instance root directory: ...

SQL Server directory: C:\Program Files\Microsoft SQL Server\MSSQL11.SQLEXPRESS

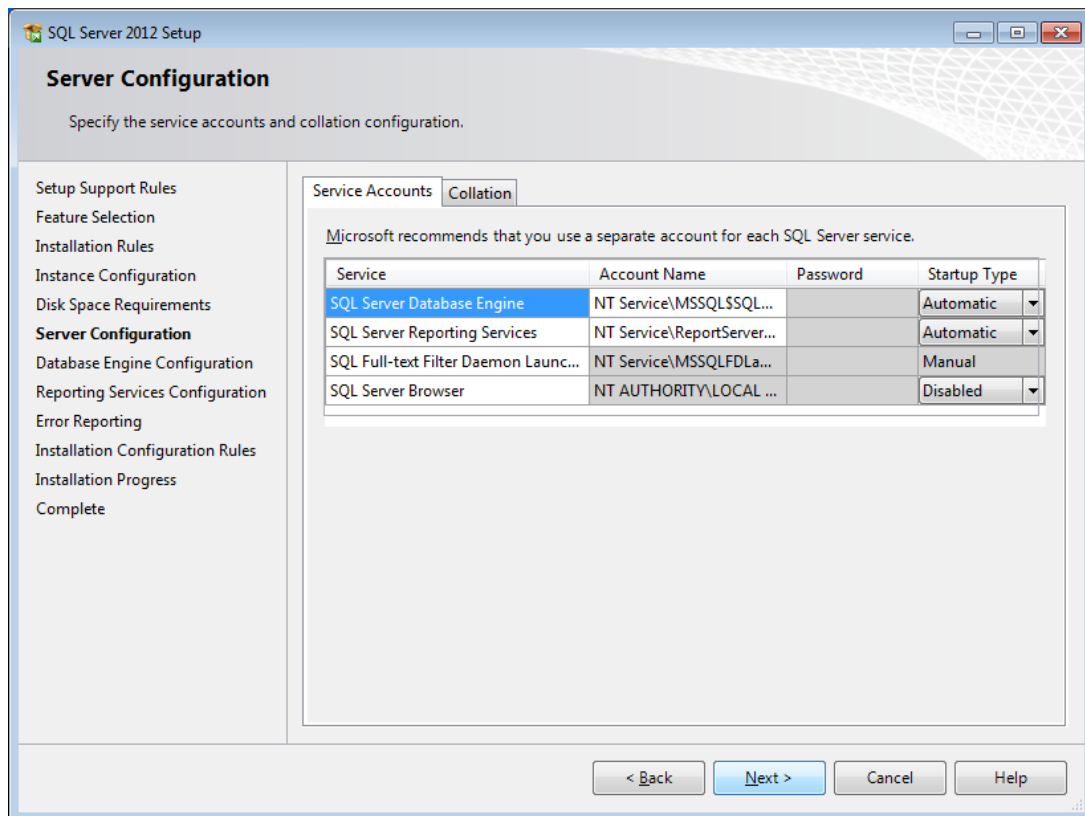
Reporting Services directory: C:\Program Files\Microsoft SQL Server\MSRS11.SQLEXPRESS

Installed instances:

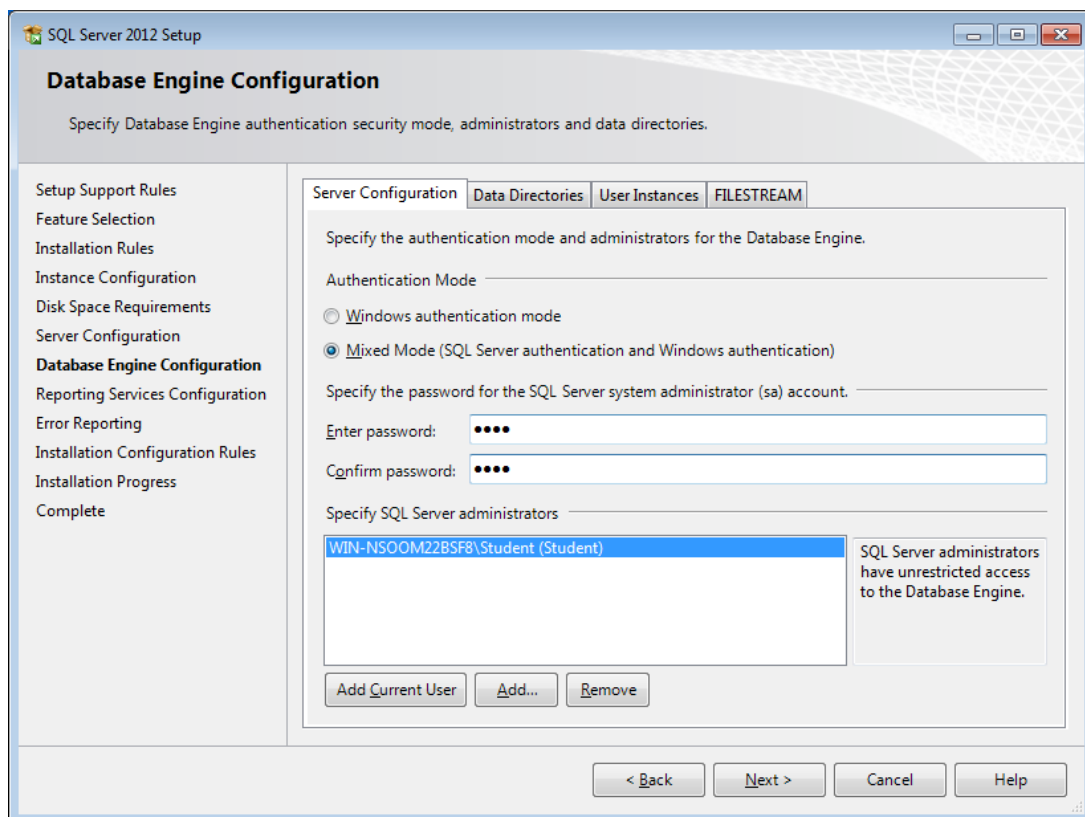
Instance Name	Instance ID	Features	Edition	Version
---------------	-------------	----------	---------	---------

< Back Next > Cancel Help

☞ Druk op de knop **Next** bij het venster **Server Configuration**. Standaard staan alle instellingen juist geconfigureerd.



Je komt nu in het venster van de **Database Engine Configuration**. In dit venster moet een aantal instellingen gewijzigd worden.

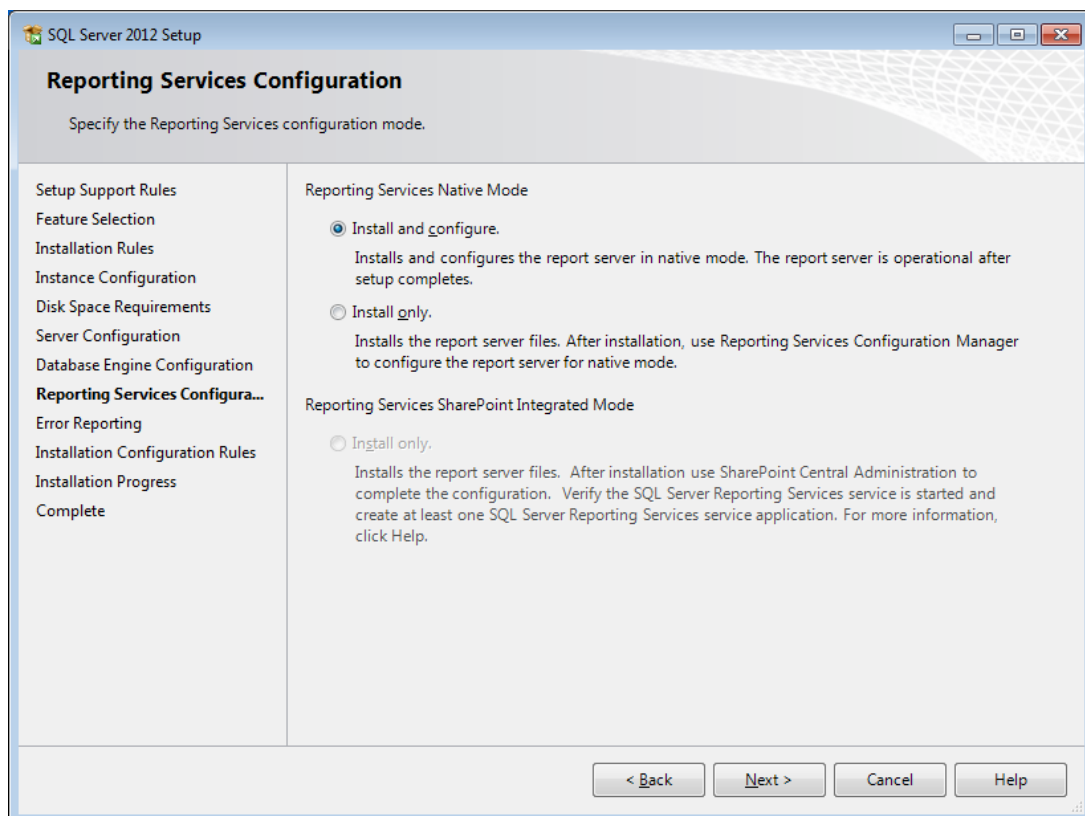


- ✓ Vink de optie **Mixed Mode** aan. Je dient nu tevens een wachtwoord op te geven. In MS SQL heb je, net als in Windows, een zogenoemde administrator. In SQL heeft deze de naam SA (System Administrator).

Als je inlogt als SA, dan mag je alles op de databaseserver. Het is dan ook van groot belang dat je dit wachtwoord niet vergeet.

- ✓ Druk op **Next**.

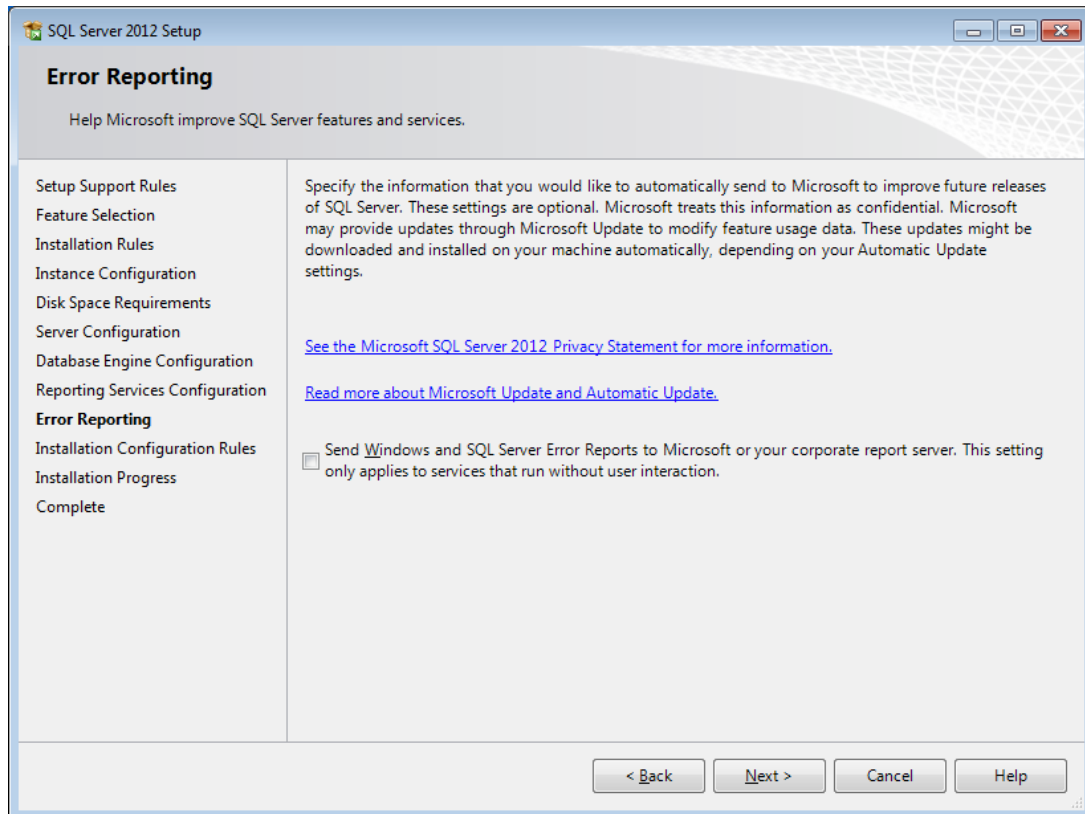
Het venster **Reporting Services Configuration** verschijnt nu. Binnen SQL Express is er een mogelijkheid om rapporten te genereren. Je kunt dit doen met de **Reporting Services**.



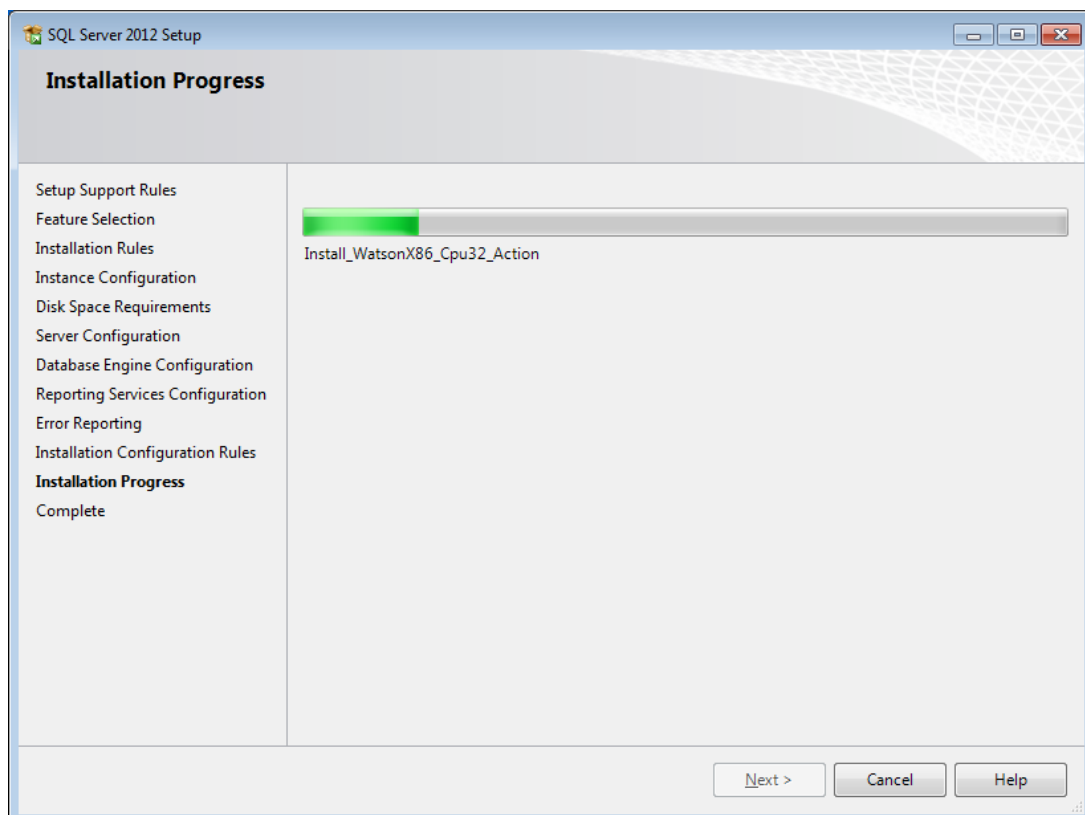
- ✓ Kies de optie **Install and configure** en druk op **Next**.

Je kunt ervoor kiezen bij het volgende scherm om wel of niet bepaalde **error reports** automatisch naar Microsoft te laten verzenden. Vink deze optie niet aan.

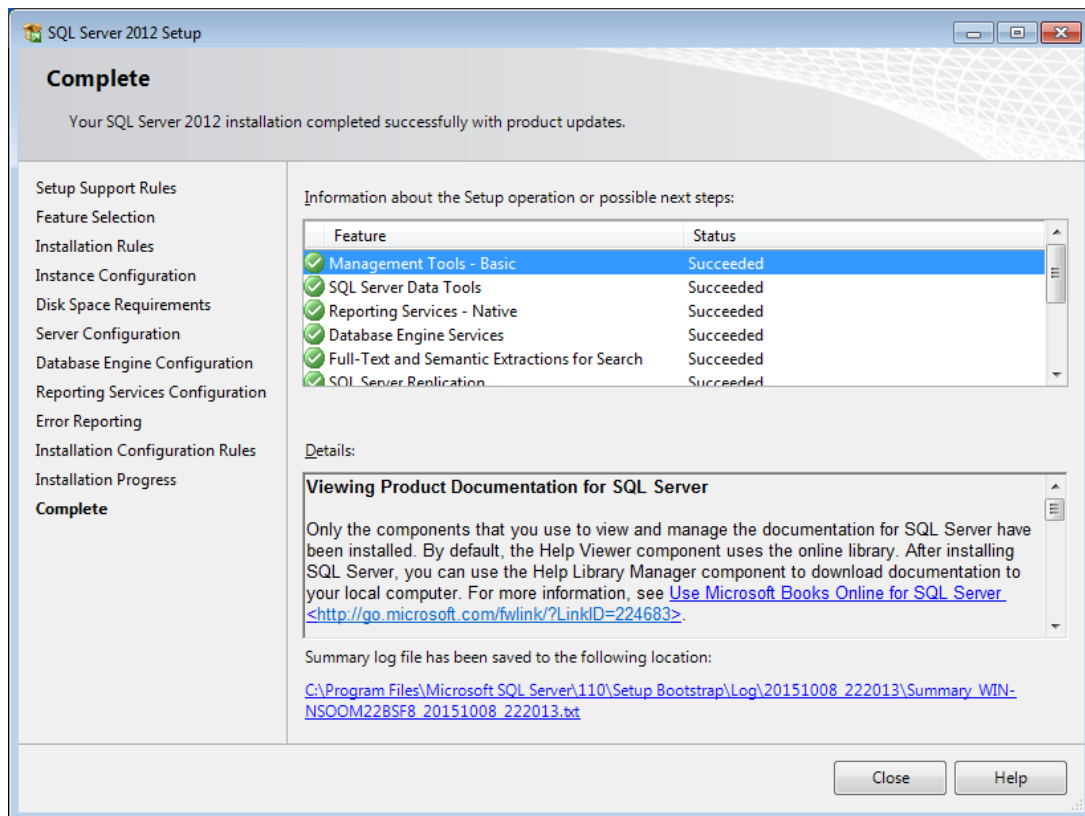
- ✓ Klik op **Next**.



Nu gaat het eigenlijke installatieproces pas van start. Dit kan enige tijd duren.



Na verloop van tijd verschijnt het laatste scherm van het installatieproces. Als alles goed is gegaan, krijg je te zien dat de installatie succesvol is verlopen.



☞ Druk op **Close** en sluit alle overige installatieschermen af.

SQL is nu geïnstalleerd. Voor je het echter kunt gebruiken, moet er nog een aantal instellingen worden aangepast. Op dit moment is de SQL-server niet van buitenaf benaderbaar. Dit is echter wel noodzakelijk.

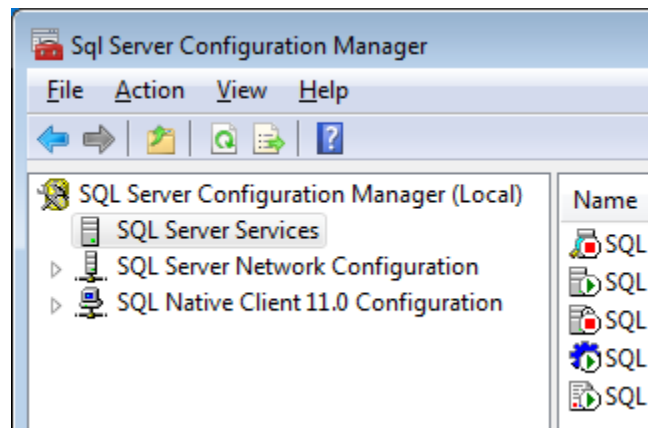
4.3 Configuratie van SQL Express

De SQL-server is nog niet van buitenaf te benaderen. Daarom moeten er twee instellingen worden aangepast. Deze aanpassingen moet je doen door gebruik te maken van de applicatie **SQL Server Configuration Manager**.

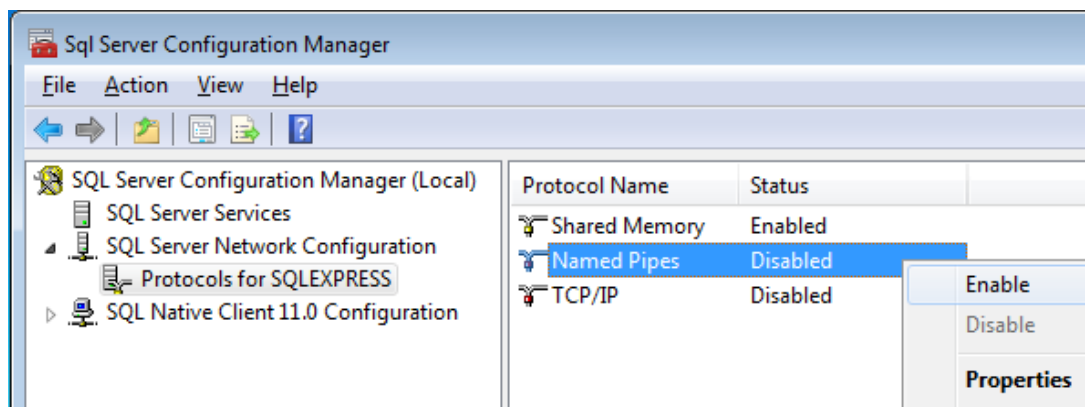
☞ Start de **SQL Server Configuration Manager** op.

Je krijgt het startscherm van **de Configuration Manager** te zien.





- ☞ Ga naar de **Protocols for SQLEXPRESS** onder het kopje **SQL Server Network Configuration**. Je krijgt de onderstaande lijst te zien.



- ☞ **Enable** de protocollen **Named Pipes** en **TCP/IP**.

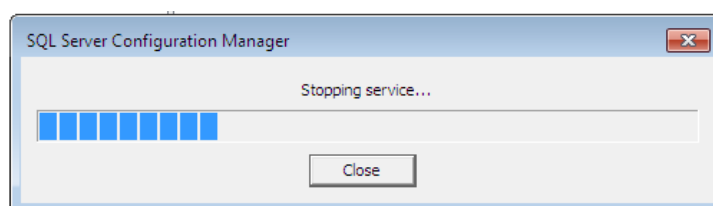
Je krijgt een melding dat de aanpassing pas van kracht wordt indien de service opnieuw gestart is.

LET OP!

Zorg ervoor dat je geen andere schermen van SQL Express meer geopend hebt voor je de service herstart.

- ☞ Ga naar de optie **SQL Server Services** en **Restart** de service **SQL Server (SQLExpress)**.

De service wordt nu herstart.



Je bent nu klaar met de configuratie.

Sluit de Configuration Manager af.

Alles is geconfigureerd.

Je kunt SQL Express gebruiken.

5 De oefendatabase

5.1 Inleiding

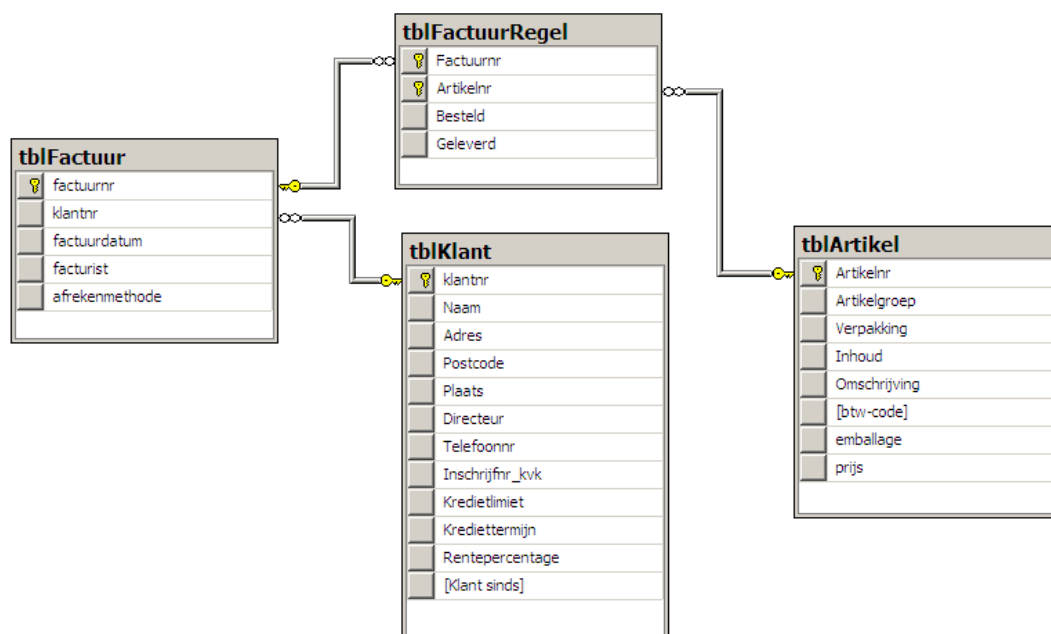
In de volgende hoofdstukken oefen en werk je met SQL. Om te kunnen oefenen heb je natuurlijk wel een database nodig. In dit hoofdstuk wordt besproken hoe je deze database op jouw computer kunt installeren. Hiervoor moet je wel SQL Server Express 2012 geïnstalleerd hebben (zie hoofdstuk 4).

Na bestudering van dit hoofdstuk, moet je tot het volgende in staat zijn.

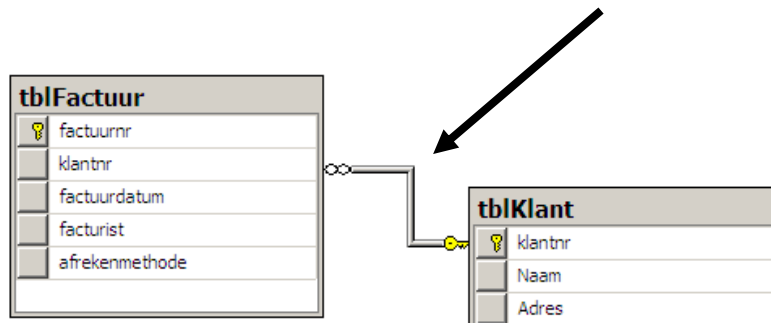
- Het kunnen uitvoeren van een SQL-script.
- Het kunnen lezen van een CREATE-script en in grote lijnen kunnen aangeven wat de functie is van de diverse statements.

5.2 De database

Hieronder is de indeling en het relatieschema weergegeven van de database.



De database bestaat uit vier tabellen die onderling een bepaalde relatie hebben. Zo kan een klant een of meer facturen hebben. Je kunt dit zien aan de relatie tussen de tabellen *tblKlant* en *tblFactuur*.



5.3 Het maken van de oefendatabase

Je weet nu hoe de database eruit gaat zien. Deze moet nog gemaakt worden. Op het netwerk staat het script ***Maken database.sql***. Dit is een zogenoemd SQL-script. In dit script staan allemaal SQL-statements die ervoor zorgen dat op jouw databaseomgeving de database *Klanten* wordt gegenereerd.

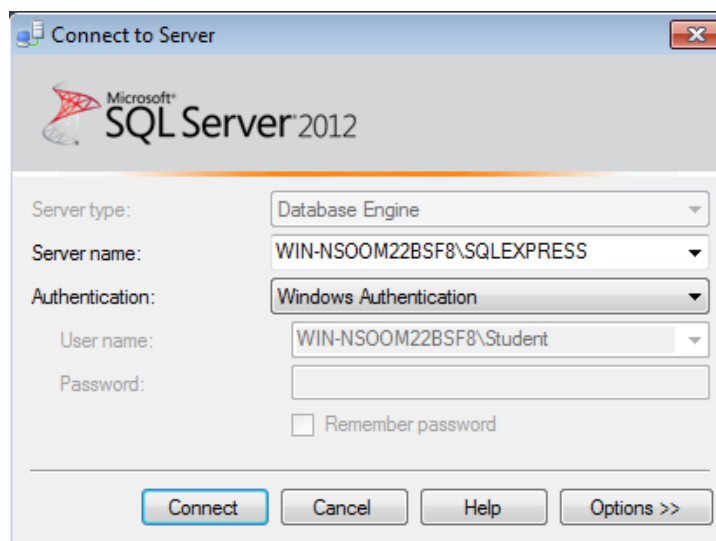
🖱 Download het script van het netwerk en plaats dit op je eigen pc.

LET OP!

Zorg ervoor dat je altijd SQL Management Studio opstart als administrator in verband met de benodigde rechten.

🖱 Open ***Microsoft SQL Server Management Studio Express***.

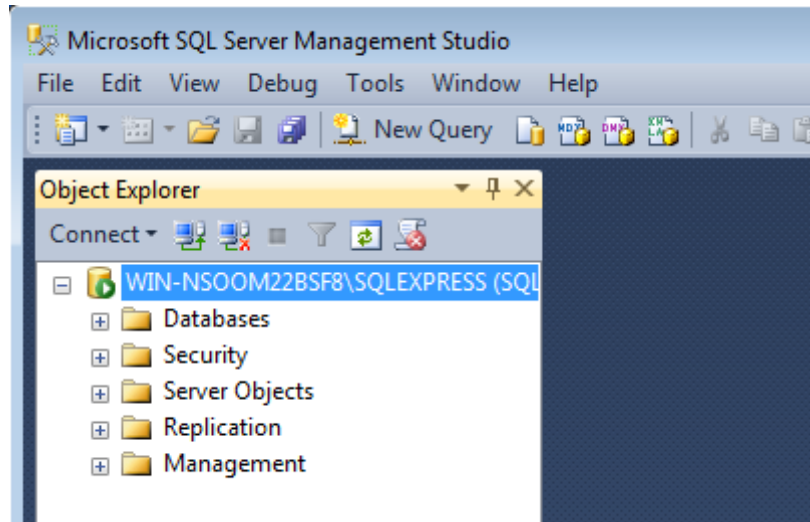
Je krijgt het onderstaande scherm te zien.



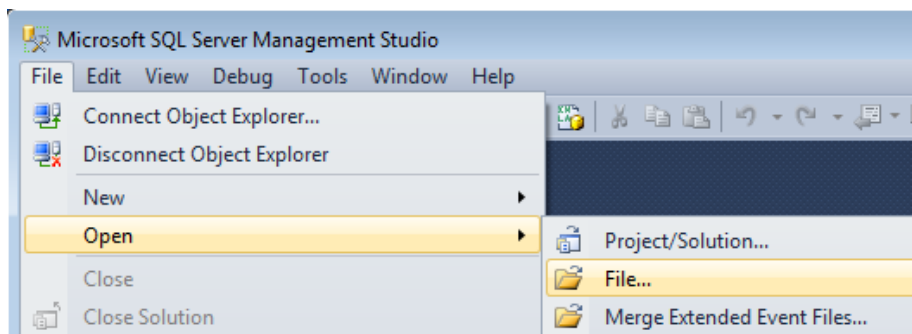
🖱 Zorg ervoor dat de ***Authentication mode*** staat op ***Windows Authentication***.

☞ Druk op **Connect**.

Je komt in het beginscherm van de Management Studio.

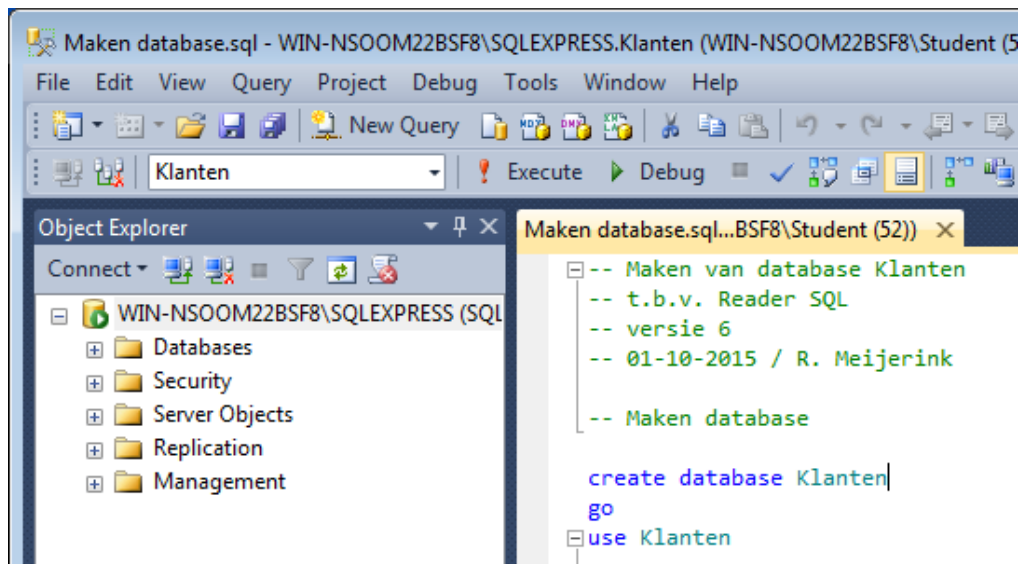


☞ Kies de opties **File, Open, File**.

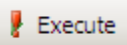


☞ Open het gedownloadde scriptbestand **Maken database.sql**.

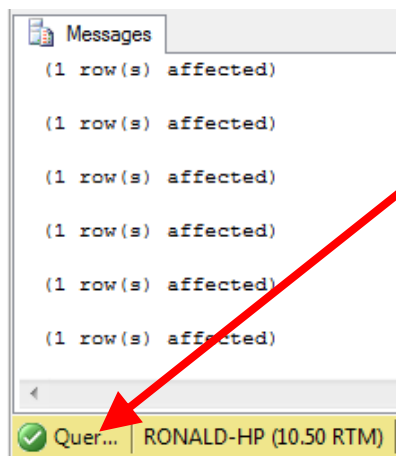
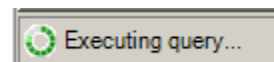
Je krijgt vervolgens het script te zien in de Management Studio.



Druk op de knop **Execute**.



Het script wordt uitgevoerd en je krijgt een aantal meldingen te zien. Als alles goed is verlopen, krijg je de melding dat alles goed is gegaan.

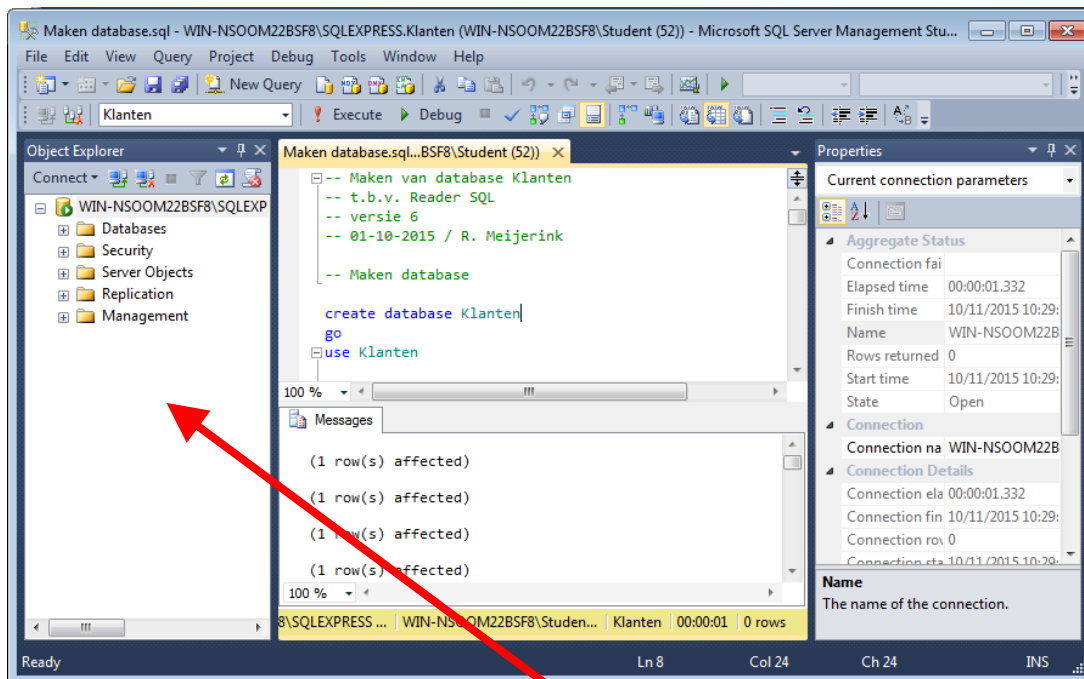


Je kunt de database gaan gebruiken.

5.4 De database bekijken

Je hebt een SQL-script uitgevoerd dat de database *Klanten* heeft gemaakt. Er is zelfs een melding verschenen dat alles succesvol is verlopen. Maar hoe kan je de gemaakte database nu zichtbaar maken?

In het volgende plaatje zie je het scherm van de Management Studio.



Aan de linkerkant staat de zogenoemde **Object Explorer**. In het rechterdeel van het scherm krijg je de detailinformatie te zien van het onderdeel dat in de Object Explorer is gekozen.

☞ Klik op het plusje voor het item **Databases**.

☞ Klik dan op het plusje (+) voor de database *Klanten*.

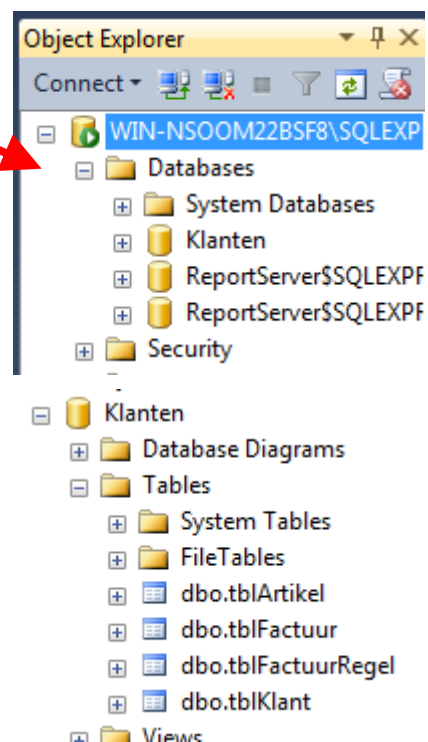
Je krijgt te zien uit welke objecten je database bestaat. De verschillende onderdelen komen in de loop van dit lesmateriaal aan bod.

☞ Klik op het plusje voor **Tables**.

Je krijgt de verschillende tabellen te zien waaruit de database bestaat.

Je hebt gezien dat de database *Klanten* is gemaakt en dat in die database vier tabellen aanwezig zijn.

Later in dit lesmateriaal ga je aan de slag met deze database.



5.5 Wat staat er in het script?

Je hebt een database met tabellen en velden. Dit is bereikt door een SQL-script uit te voeren. In paragraaf 5.3 heb je al kunnen lezen dat een SQL-script niets anders is dan een aantal SQL-statements achter elkaar. Deze SQL-statements hebben ervoor gezorgd dat de database is gemaakt.

In het begin van het script staan de onderstaande regels.

De groene regels zijn zogenaamd commentaar. Dit zijn opmerkingen die de schrijver van het SQL-Script heeft gemaakt.

Met het statement **CREATE** kun je objecten aanmaken in SQL. In dit geval maak je een database met de naam *Klanten*.

In SQL moet je, voor je een object gaat gebruiken, dit definitief maken. Dit doe je door het statement **GO** uit te voeren.

Hierna moet je alleen nog aangeven dat je deze database ook wilt gaan gebruiken. Dit doe je door het statement **USE Klanten**.

```
-- Maken van database Klanten
-- t.b.v. Reader SQL
-- versie 6
-- 01-10-2015 / R. Meijerink

-- Maken database
```

```
create database Klanten
go
use Klanten
```

Je kunt de database *Klanten* nu gaan gebruiken. In deze database ga je een tabel aanmaken.

Dit doe je ook weer door gebruik te maken van het statement **CREATE**. Nu maak je alleen een **table** (= tabel) aan met de naam *tblKlant*.

Na het aanmaken van de tabel, komen alle velden die in het tabel zitten. Achter deze velden staat het gegevenstype. Hier vul je in wat voor gegevens in het veld mogen staan.

Als laatste wordt de primaire sleutel van de tabel aangemaakt.

```
-- Maken tabellen database

create table tblKlant (
    klantnr          char (5),
    Naam             varchar (30),
    Adres            varchar (30),
    Postcode         char (6),
    Plaats           varchar (20),
    Directeur        varchar (30),
    Telefoonnr       char (11),
    Inschrijfnr_kvkv char (10),
    Kredietlimiet     int,
    Krediettermijn    int,
    Rentepercentage  int,
    [Klant sinds]    datetime,
    primary key (Klantnr));
```

Na het aanmaken van de tabellen moeten de relaties tussen deze tabellen worden gemaakt. Dit doe je door gebruik te maken van het statement

ALTER TABLE <tabelnaam> ADD FOREIGN KEY

Als je dit statement letterlijk vertaalt, staat er het volgende.
verander tabel <tabelnaam> voeg verwijzende sleutel toe

```
-- Maken relaties tussen de tabellen
```

```
alter table tblFactuurRegel add foreign key (Factuurnr) references tblFactuur (Factuurnr);
alter table tblFactuurRegel add foreign key (Artikelnr) references tblArtikel (Artikelnr);
alter table tblFactuur add foreign key (Klantnr) references tblKlant (Klantnr);
```

Het laatste onderdeel van het script is het vullen van de tabellen met gegevens. Dit doe je door gebruik te maken van het statement

INSERT INTO <tabelnaam> VALUES (<gegevens>).



```
-- Vullen van de tabellen
```

```
insert into tblArtikel values ('DVD','doos',24,'Ice age III',2,0,19.95)
insert into tblArtikel values ('BLUERAY','doos',24,'Ice age III',2,0,29.95)
insert into tblArtikel values ('DVD','doos',24,'Star Wars 2',2,0,11.95)
insert into tblArtikel values ('CD','Spindel',50,'TDK CR-R 700mb',2,0,14.95)
```

Je hebt gezien wat er allemaal in het script staat. Alle genoemde statements komen uitgebreid aan bod in het vervolg van dit lesmateriaal.

6 Het maken van een database

6.1 Inleiding

In hoofdstuk 5 heb je een oefendatabase gemaakt door middel van een script. In dit hoofdstuk ga je zelf een database maken en deze vullen met tabellen. Hiervoor moet je gebruik gaan maken van DDL-statements.

In de Data Definition Language (DDL) kun je onder meer nieuwe tabellen creëren, de structuur van bestaande tabellen wijzigen en tabellen verwijderen. Met andere woorden: alles wat nodig is om een database te maken.

Na bestudering van dit hoofdstuk moet je tot het volgende in staat zijn.

- Het kunnen maken van een CREATE-script.
- Het kunnen werken met gegevenstypes en hierbij de juiste types selecteren.
- Het achteraf kunnen toevoegen en verwijderen van sleutels.
- Het kunnen maken van een databasediagram.

6.2 Syntax

Wat volgt, is een syntaxoverzicht van de gebruikte statements in dit hoofdstuk.

Maken van een database

```
CREATE DATABASE <Naam>
```

Maken van een tabel met primaire sleutel

```
CREATE TABLE <Tabelnaam>(  
<veld1>          <Gegevenstype>,  
<veld2>          <Gegevenstype>,  
<veld..>         <Gegevenstype>,  
CONSTRAINT <Sleutelnaam> PRIMARY KEY CLUSTERED(<veld 1>,  
<veld 2>,<veld ..>))
```

Het verwijderen van een database

```
DROP DATABASE <databasenaam>
```

Achteraf toevoegen van een primaire sleutel

```
ALTER TABLE <tabelnaam>  
ADD CONSTRAINT <Sleutelnaam>  
PRIMARY KEY (<veld 1>,<veld 2>,<veld ..>)
```

Verwijderen van een sleutel

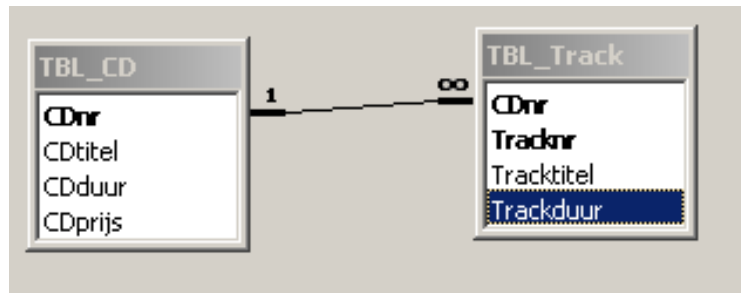
```
ALTER TABLE <tabelnaam>  
DROP CONSTRAINT <sleutelnaam>
```

Het achteraf toevoegen van een foreign key (= verwijzende sleutel)

```
ALTER TABLE <tabelnaam>  
ADD CONSTRAINT <Sleutelnaam>  
FOREIGN KEY (<veld 1>,<veld ..>)  
REFERENCES <tabelnaam> (<veld 1>,<veld ..>)
```

6.3 Voorbeelddatabase

In dit hoofdstuk maak je onderstaande database in SQL.



De database bestaat uit twee tabellen die een relatie met elkaar hebben. De volgende objecten moet je creëren.

- Database
- Twee tabellen
- Primaire sleutels
- Verwijzende sleutel

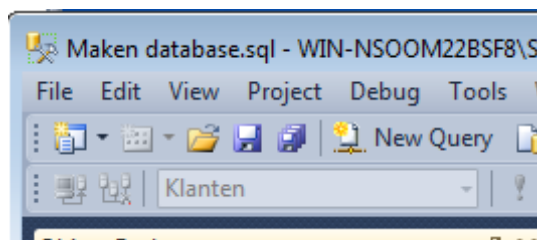
6.4 Het maken van een database


Voor je een tabel kunt aanmaken, moet de database zelf aangemaakt worden. Om een database aan te maken, gebruik je de onderstaande syntax.

```
CREATE DATABASE <databasenaam>
```

Je gaat een database aanmaken om een cd-verzameling te kunnen bijhouden.

🖱️ Open een nieuwe query.



 Neem de volgende code in het queryvenster

```
create database CD  
go  
use CD
```

In het bovenstaande statement wordt de database *CD* gemaakt. Nu kan er een begin gemaakt worden met het aanmaken van tabellen in de database.

6.5 Het maken van een tabel

Voor het maken van tabellen gebruik je het DDL-statement CREATE TABLE. De syntax van het statement is als volgt.

```
CREATE TABLE <tabelnaam>(  
<veld1>      <gegevenstype>,  
<veld2>      <gegevenstype>,  
<veld...>    <gegevenstype>)
```

De syntax die hierboven staat, is de meest eenvoudige vorm. Later in dit lesmateriaal zal de volledige syntax met voorbeelden aan bod komen.

Bij gegevenstype moeten we invullen wat voor gegevens in het veld mogen staan. In de tabel op de volgende pagina staat een aantal bij de meeste databaseomgevingen te gebruiken gegevens. Voor een compleet overzicht van gegevenstypes dient de handleiding van de databaseomgeving geraadpleegd te worden.

<http://msdn.microsoft.com/en-us/library/ms187752.aspx>

Gegevenstype	Omschrijving
Bigint	-2 ⁶³ (-9,223,372,036,854,775,808) to 2 ⁶³ -1 (9,223,372,036,854,775,807)
Int	-2 ³¹ (-2,147,483,648) to 2 ³¹ -1 (2,147,483,647)
Smallint	-2 ¹⁵ (-32,768) to 2 ¹⁵ -1 (32,767)
Tinyint	0 to 255
Bit	1, 0 of NULL
Decimal	38 teken achter de punt
Numeric	Zelfde als Decimal
Money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
smallmoney	- 214,748.3648 to 214,748.3647
Float	- 1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308 Float(24) is hetzelfde als Real
Real	- 3.40E + 38 to -1.18E - 38, 0 and 1.18E - 38 to 3.40E + 38
Datetime	January 1, 1753, through December 31, 9999
Smalldatetime	January 1, 1900, through June 6, 2079
Date	01-01-0001 through 31-12-9999
Time	00:00:00.0000000 through 23:59:59.9999999
Char(x)	Non-unicode (waar x het aantal tekens is.) Maximaal 8000
Varchar(x)	Non-unicode (waar x het aantal tekens is.) Maximaal 8000
Text	2,147,483,647 tekens in non-unicode
Nchar(x)	Unicode (waar x het aantal tekens is.) Maximaal 4000
Nvarchar(x)	Unicode (waar x het aantal tekens is.) Maximaal 4000
Ntext	1,073,741,823 tekens in unicode
Binary(x)	Binair (waar x het aantal bytes is.) Maximaal 8000
Varbinary(x)	Binair (waar x het aantal bytes is.) Maximaal 8000
Image	Binary data maximaal 2,147,483,647 bytes

Char versus Varchar

De omschrijvingen van de gegevenstypes Char en Varchar zijn deze exact hetzelfde. Toch is er een belangrijk verschil tussen de twee. Dit verschil wordt duidelijk als je kijkt naar het onderstaand voorbeeld.

Het veld *Plaats* wordt aangemaakt met als gegevenstype Char(30) of Varchar(30). In het veld *Plaats* wordt de tekst 'Boxtel' gezet.

Char^{)*} *Boxtel* _ _ _ _ _

Varchar^{)*} *Boxtel* _ _


^{)*} _ = bezet geheugenruimte

Je ziet dat bij het gegevenstype Char de lege plaatsen worden opgevuld met spaties. Bij het gegevenstype Varchar gebeurt dit niet met alle plaatsen.

Hieruit is op te maken dat het gegevenstype Char gebruikt moet worden bij velden met een vaste lengte, zoals een postcode of een telefoonnummer.

Bij velden waar de lengte variabel is, moet het gegevenstype Varchar gebruikt worden.


Je maakt nu de tabel *TBL_CD*.

 Ga onder de reeds aanwezige SQL-code staan en neem het onderstaande over.

```
create table TBL_CD(
CDnr          int,
CDtitel       varchar(40),
CDduur        real,
CDprijs       money)
```

 Voer het script uit door op  te klikken. Je kunt ook de **F5**-toets gebruiken.

De database *CD* met daarin de tabel *TBL_CD* wordt gemaakt.

 Bekijk in de **Object Explorer** of de database is gemaakt en of de tabel *TBL_CD* aanwezig is. Het kan zijn dat je eerst op **Refresh** moet klikken om de database zichtbaar te maken.


6.6 Het wissen van een database

Het kan natuurlijk voorkomen dat een database niet meer gebruikt wordt. Je kunt er dan voor kiezen om de database leeg te maken en alsnog te bewaren of de hele database te wissen.

Voor het wissen van objecten als database, tabel of sleutel gebruik je het statement **DROP**. De syntax van het statement is als volgt.

```
DROP DATABASE <databasenaam>
```

Wis de database *CD*.

 Ga naar de reeds aanwezige SQL-code en voeg bovenaan in de code de onderstaande SQL-statements toe.

```
use master
drop database CD
go
```

- ☞ Selecteer alleen de drie toegevoegde statements in je code en druk op **F5**. Alleen de geselecteerde regels worden nu uitgevoerd.
- ☞ Kijk of de database *CD* daadwerkelijk is gewist. Vergeet niet een **refresh** te doen.

Elke keer dat je het script uitvoert, wordt de database eerst gewist en daarna opnieuw aangemaakt.

6.7 Primaire sleutel

Je bent bij het aanmaken van *TBL_CD* iets vergeten. In een relationele database heeft iedere tabel een primaire sleutel. Deze sleutel is uniek voor elk record.

In SQL zijn er drie mogelijkheden om primaire sleutels aan te maken.

Achteraf een primaire sleutel toekennen.

Om achteraf iets toe te voegen en/of aan te passen gebruik je het statement **ALTER**. Je past dadelijk iets aan in een tabel. Daarom gebruik je **ALTER TABLE**.

De syntax om achteraf een primaire sleutel toe te voegen.

```
ALTER TABLE <tabelnaam>
ADD CONSTRAINT <Sleutelnaam>
PRIMARY KEY (<veld 1>,<veld 2>,<veld ..>)
```

In het eerder getoonde Bachmandiagram is te zien dat de primaire sleutel van de tabel *TBL_CD* het veld *CDnr* moet worden. Het statement ziet er dan als volgt uit.

```
ALTER TABLE TBL_CD
ADD CONSTRAINT PR_CDnr
PRIMARY KEY (CDnr)
```

Een primaire sleutel toevoegen tijdens het aanmaken van een tabel.

De tweede methode om een primaire sleutel toe te voegen aan een tabel is te gebruiken bij het CREATE TABLE-statement. De syntax is hieronder weergegeven.

```
CREATE TABLE <tabelnaam>(
<veld1>      <gegevenstype>,
<veld2>      <gegevenstype>,
<veld...>    <gegevenstype>,
PRIMARY KEY (<veld 1>,<veld 2>,<veld ..>))
```

Om de tabel *TBL_CD* aan te maken inclusief primaire sleutel moet het statement ingegeven worden dat hieronder staat weergegeven.

```
CREATE TABLE TBL_CD(  
  CDnr          INT,  
  CDtitel       VARCHAR(40),  
  CDduur        REAL,  
  CDprijs       REAL,  
  PRIMARY KEY (CDnr))
```

Tijdens het aanmaken van een tabel een primaire sleutel toevoegen en een naam geven

De derde methode om een primaire sleutel toe te voegen is tevens de meest gebruikte methode. Op deze manier kan een ontwikkelaar elke sleutel gelijk een naam geven. Deze methode moet je evenals de tweede methode toepassen tijdens het CREATE TABLE-statement. De syntax is hieronder weergegeven.

```
CREATE TABLE <tabelnaam>(  
  <veld1>       <gegevenstype>,  
  <veld2>       <gegevenstype>,  
  <veld...>     <gegevenstype>,  
  CONSTRAINT <sleutelnaam> PRIMARY KEY CLUSTERED(<veld1>))
```

Om de tabel *TBL_CD* aan te maken inclusief de primaire sleutel met als naam *PR_CDnr* moet het onderstaande statement worden ingegeven.

```
CREATE TABLE TBL_CD(  
  CDnr          INT,  
  CDtitel VARCHAR(40),  
  CDduur        REAL,  
  CDprijs REAL,  
  CONSTRAINT PR_CDnr PRIMARY KEY CLUSTERED(CDnr))
```

☞ Pas de SQL-code zo aan dat de tabel *TBL_CD* een primaire sleutel krijgt.

Je mag zelf kiezen welke methode je gebruikt.

☞ Voer het script opnieuw uit om de database met de tabel te maken.

6.8 Het verwijderen van een primaire sleutel

Je hebt al kennis gemaakt met het statement DROP. Ook als je een primaire sleutel verwijdert, moet je dit statement gebruiken. Het verwijderen van een primaire sleutel is echter wel wat lastiger. Je moet namelijk de naam van de betreffende primaire sleutel kennen.

De syntax is als volgt.

```
ALTER TABLE <tabelnaam>
DROP CONSTRAINT <sleutelnaam>
```

Om de primaire sleutel van de tabel TBL_CD te verwijderen gebruik je het onderstaande statement.


```
ALTER TABLE TBL_CD
DROP CONSTRAINT prCDnr
```

Let er wel op dat alleen de sleutel wordt verwijderd! Het veld *CDnr* blijft gewoon aanwezig in *TBL_CD*.

6.9 De verwijzende sleutel


Naast de primaire sleutels kom je in de database ook verwijzende sleutels (foreign keys) tegen. Deze sleutels worden gebruikt om de relaties tussen de diversen tabellen te maken of weer te geven.

Om echter een verwijzende sleutel te maken, moet je eerst twee tabellen hebben. Je maakt daarom eerst de tabel *TBL_Track*.

 Voeg onderaan in je SQL-script het onderstaande SQL-statement toe.

```
create table TBL_Track(
  CDnr                int,
  Tracknr             int,
  Tracktitel          varchar(80),
  Trackduur           int,
  CONSTRAINT PR_Track PRIMARY KEY CLUSTERED (CDnr, Tracknr))
```

In het bovenstaande statement wordt tijdens het maken van de tabel tevens een primaire sleutel aangemaakt. Het gaat hier om een samengestelde sleutel en daarom moet je achter de optie CLUSTERED de twee sleutelvelden aangeven.

 Voer het script uit om de database te maken.

Het maken van een verwijzende sleutel is altijd lastiger dan het aanmaken van een primaire sleutel. Voor een verwijzende sleutel heb je immers altijd twee tabellen nodig.

Het is mogelijk om tijdens het aanmaken van een tabel een verwijzende sleutel toe te voegen. Je moet dan echter rekening houden met het databaseontwerp. Het is niet mogelijk om een verwijzende sleutel toe te voegen tussen twee tabellen als een van deze tabellen niet aanwezig is.

In de praktijk houdt dit in dat het toevoegen van een verwijzende sleutel nagenoeg altijd achteraf zal plaatsvinden. We gebruiken hiervoor de volgende syntax.

```
ALTER TABLE <tabelnaam>  
ADD CONSTRAINT <sleutelnaam>  
FOREIGN KEY (<veld 1>,<veld ..>) REFERENCES <tabelnaam> (<veld 1>,<veld ..>)
```

Je legt de relatie tussen *TBL_CD* en *TBL_Track*.

☞ Ga aan het einde van je script staan en neem de onderstaande SQL-code over.

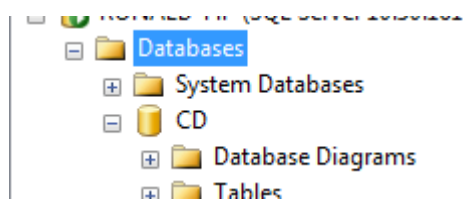
```
alter table TBL_Track  
add constraint FK_CDTrack  
foreign key (CDnr) references TBL_CD (CDnr)
```

☞ Voer nogmaals het complete script uit.

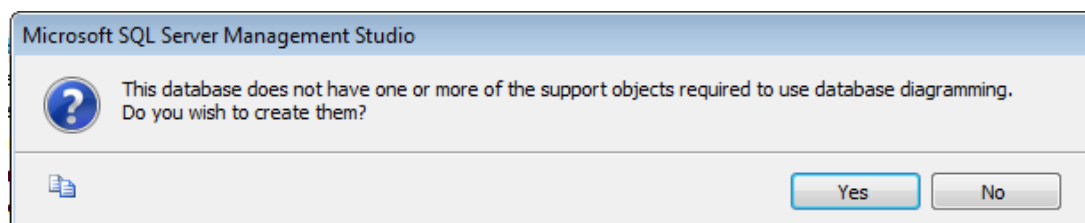
De database *CD* wordt opnieuw aangemaakt met een relatie tussen twee tabellen. Je wilt dit natuurlijk wel kunnen controleren. Hiervoor moet je in SQL een Database Diagram (= Bachmandiagram) laten genereren.

☞ Ga naar de **Object Explorer**. Zorg er wel voor dat je eerst een refresh hebt uitgevoerd.

☞ Open de database *CD* en klik op het plusje voor **Database Diagrams**.

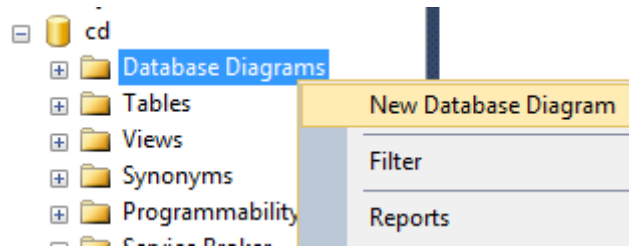


Je krijgt de volgende melding, omdat er nog geen diagrammen aanwezig zijn in deze database.

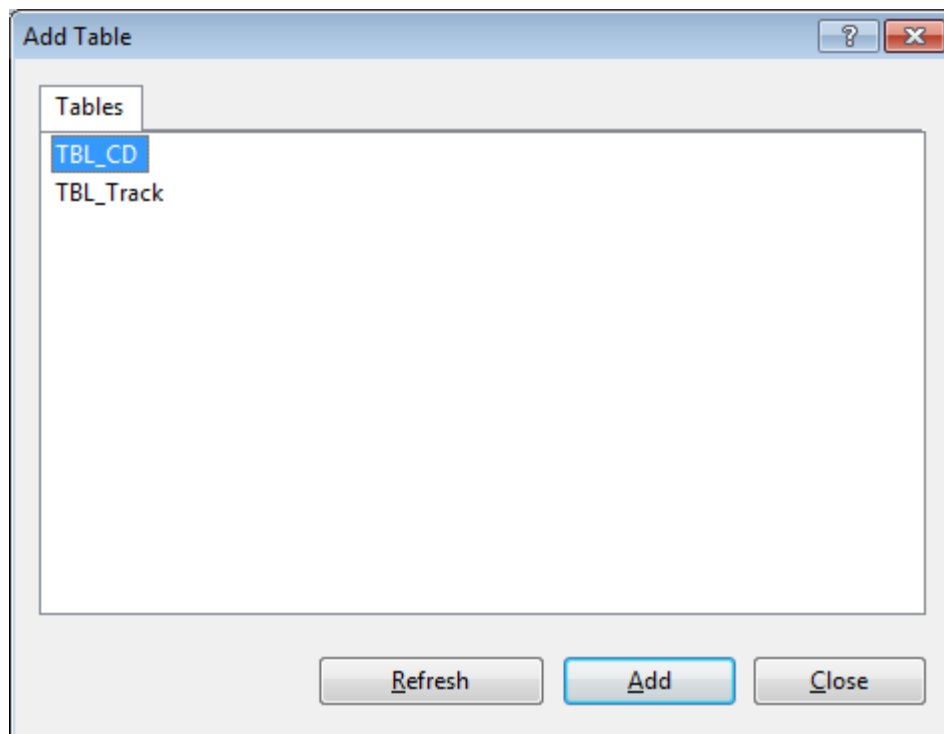


☞ Klik op **Yes**.

- ☞ Ga nu met je muis op **Database Diagrams** staan en druk op de rechtermuisknop.
- ☞ Kies dan de optie **New Database Diagram**.




Je krijgt het volgende scherm te zien.



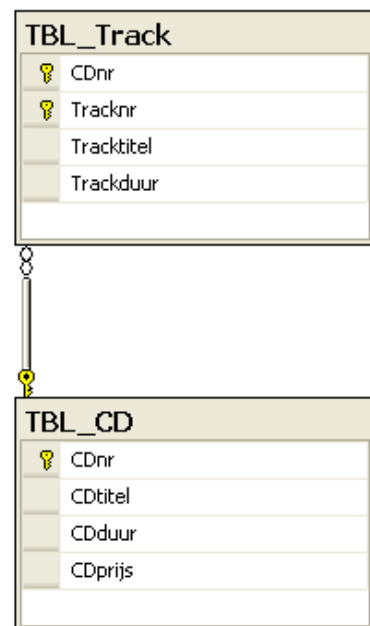
- ☞ Selecteer alle tabellen en druk op **Add**.

SQL Express Management Studio maakt nu op de achtergrond een databasediagram.

 Druk op **Close**.

Zoals je ziet is de relatie gelukt.

In tegenstelling tot MS Access kun je in dit databasediagram niet zien welke velden tot de verwijzende sleutel behoren. Daarnaast wordt i.p.v. het cijfer '1' de sleutel gebruikt.



LET OP!

Een verwijzende sleutel moet altijd naar een uniek veld verwijzen.
In vrijwel alle gevallen is dit de primaire sleutel.

6.10 Samenvatting

In dit hoofdstuk heb je gezien hoe je een database kan maken door gebruik te maken van een SQL-script. Je hebt een eenvoudige database gemaakt, die bestaat uit twee tabellen met een relatie tussen elkaar.

Alle SQL-statements die je hebt gebruikt, behoren tot de DDL-groep (Data Definition Language). Er is echter nog veel meer mogelijk, zoals werken met indexen of in bestaande tabellen velden toevoegen, verwijderen en hernoemen. Deze zaken komen later in deze reader pas aan bod.

Alle stappen, die je hebt gemaakt om een database te genereren, zijn behalve in code ook grafisch uit te voeren. Toch zal dit bij het maken van een database niet vaak voorkomen. Een database wordt ontwikkeld en tijdens het testen ervan komen er altijd onderdelen naar boven die net even anders moeten. In een script is dit eenvoudig aan te passen.

De database wordt verwijderd en het script wordt opnieuw uitgevoerd en er is binnen enkele seconden weer een werkbare database. Als je dit grafisch zou doen, moet je alle stappen weer een voor een uitvoeren om de database te maken.

LET OP!

Op een grafische manier objecten als databases en tabellen maken, lijkt sneller en eenvoudiger. In de praktijk blijkt echter dat dit vaak veel meer tijd kost.

Gebruik daarom liefst scripts indien mogelijk.

Maak nu opdracht 6.1 + 6.2.

7 Het selecteren van gegevens

7.1 Inleiding

Het doel van databases is het ontsluiten van gegevens. Een database wordt gebruikt om gegevens in op te slaan en deze terug te kunnen lezen. In dit hoofdstuk ga je door gebruik te maken van het SQL-statement **SELECT** kijken hoe je gegevens uit een database kunt halen.

Na bestudering van dit hoofdstuk dient de gebruiker tot het volgende in staat te zijn.

- Het kunnen toevoegen van commentaar in scriptfiles.
- Eenvoudige selecties uitvoeren op een tabel.
- Het kunnen ontdubbelen van selectieresultaten.
- Het kunnen werken met selectievoorwaarden.
- Het juist kunnen toepassen van AND, OR en NOT.

7.2 Syntax

Hieronder is een syntaxoverzicht van de gebruikte statements in dit hoofdstuk.

Het laten zien van alle velden van alle records

```
SELECT *  
FROM <tabelnaam>
```

Alleen de aangegeven velden laten zien

```
SELECT <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam>
```

Alle velden laten zien zonder dubbele records

```
SELECT DISTINCT *  
FROM <tabelnaam>
```

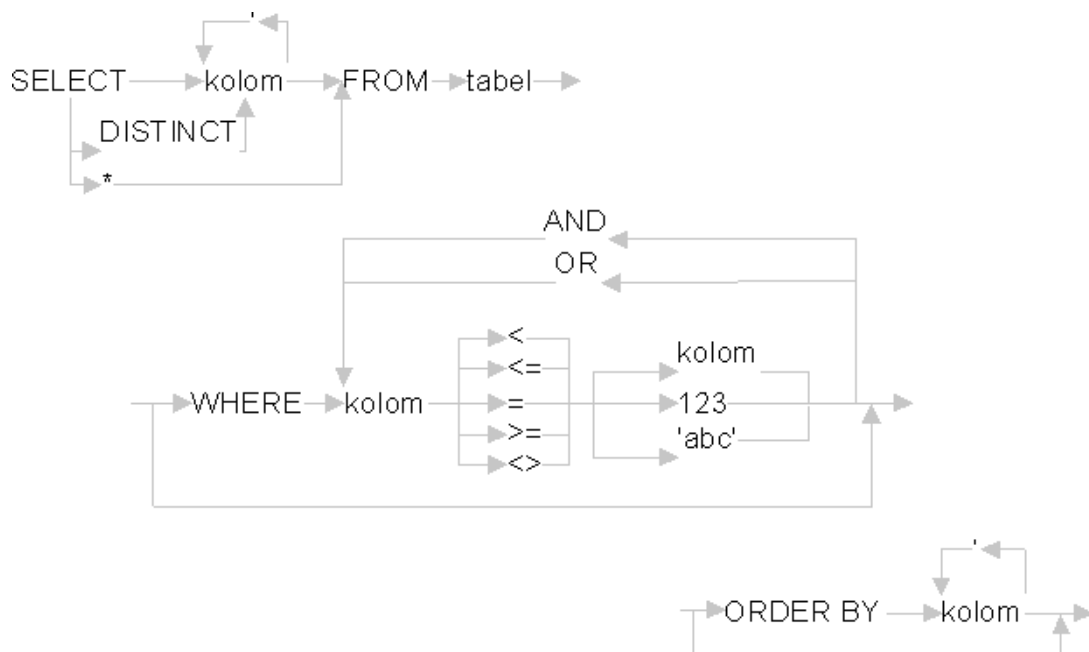
Alleen de aangegeven velden laten zien met de betreffende voorwaarden

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam>  
WHERE <veldnaam> operator <waarde>
```

7.3 SELECT

Voor het raadplegen van gegevens wordt het DML-statement SELECT gebruikt. Elk statement heeft een syntax. Dit is de schrijfwijze van het statement. Hierna staat de vereenvoudigde syntax van het select-statement weergegeven.

SELECT-statement



Het bovenstaande syntaxdiagram zal waarschijnlijk nog niet duidelijk zijn. Tijdens de lessen zal dit diagram uitvoerig besproken worden. Daarnaast zal het ook een stuk duidelijker zijn geworden aan het einde van dit hoofdstuk.

7.4 De eenvoudigste versie van SELECT

Het statement SELECT heeft (in zijn eenvoudigste vorm) de volgende syntax.

```
SELECT *
FROM <tabelnaam>
```

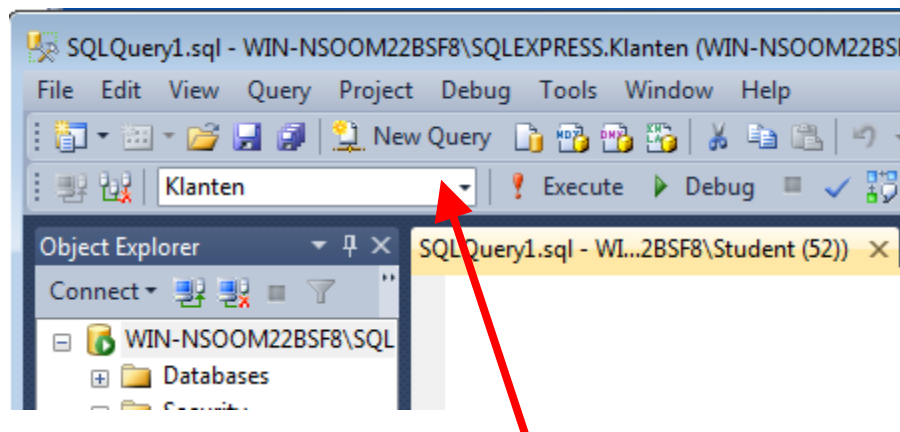
De asterisk (* - het sterretje) in de opdracht geeft aan dat alle velden van de tabel moeten worden weergegeven.

Dit ga je nu uittesten op de oefendatabase.

🔗 Ga naar de **Management Studio**.

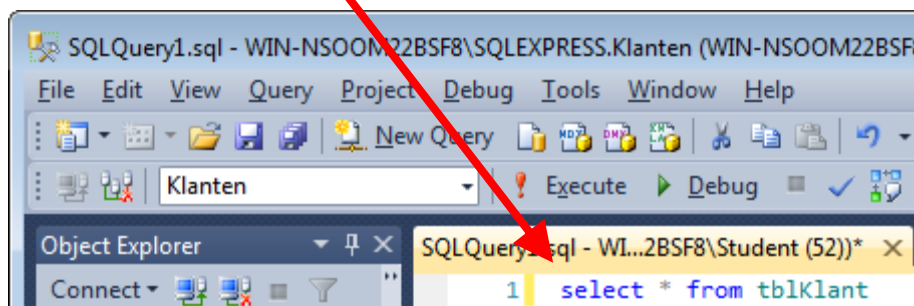
🔗 Druk op de knop **New Query** .

Er verschijnt een venster waar je SQL-statements kunt invoeren.



☞ Selecteer de database *Klanten* uit de dropdownlist. Uit deze database moeten de gegevens komen.

☞ Voer het onderstaande statement in en druk op **Execute** .



De query wordt uitgevoerd en je krijgt het volgende resultaat te zien.

	klantnr	Naam	Adres	Postcode	Plaats	Directe
1	AMC	Academisch ziekenhuis	Oosterbaan 31	1012XX	Amsterdam	A. Grie
2	AVANS	Avans Den Bosch	Onderwijsboulevard 5	5052DE	Den Bosch	L. Eerg
3	BMC	Bosch Medisch Centrum	Vijmenseweg 13	5051HT	Den Bosch	C. Ardi

Je ziet dat alle gegevens uit de tabel *tblKlant* worden weergegeven. Dit zijn gegevens van 10 klanten.

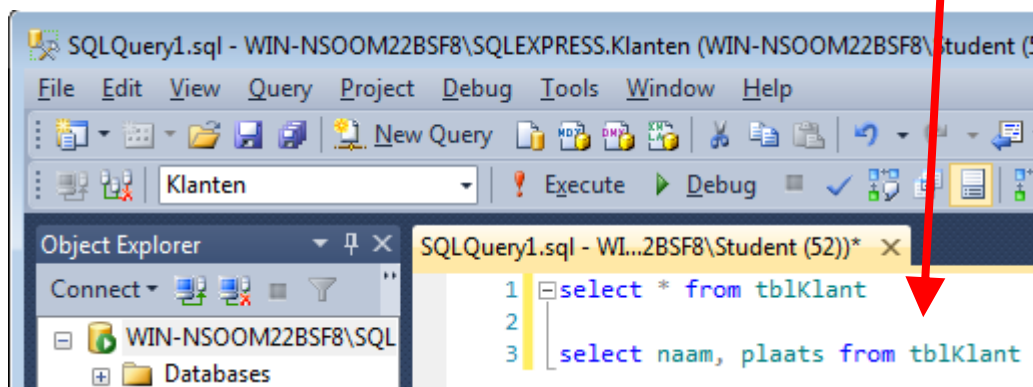
Meestal wil je echter dat niet alle velden uit een tabel worden weergegeven in een overzicht. Je wilt zelf aangeven welke velden je wel of juist niet wilt weergeven.

Dit kun je bereiken door de onderstaande syntax te gebruiken.

```
SELECT <veld1>, <veld2>, ..., <veldN>
FROM <tabelnaam>
```

Maak nu een overzicht met de namen en plaatsen van alle klanten uit de tabel *tblKlant*.

- Ga naar het queriescherm en vul hierop de onderstaande query in.



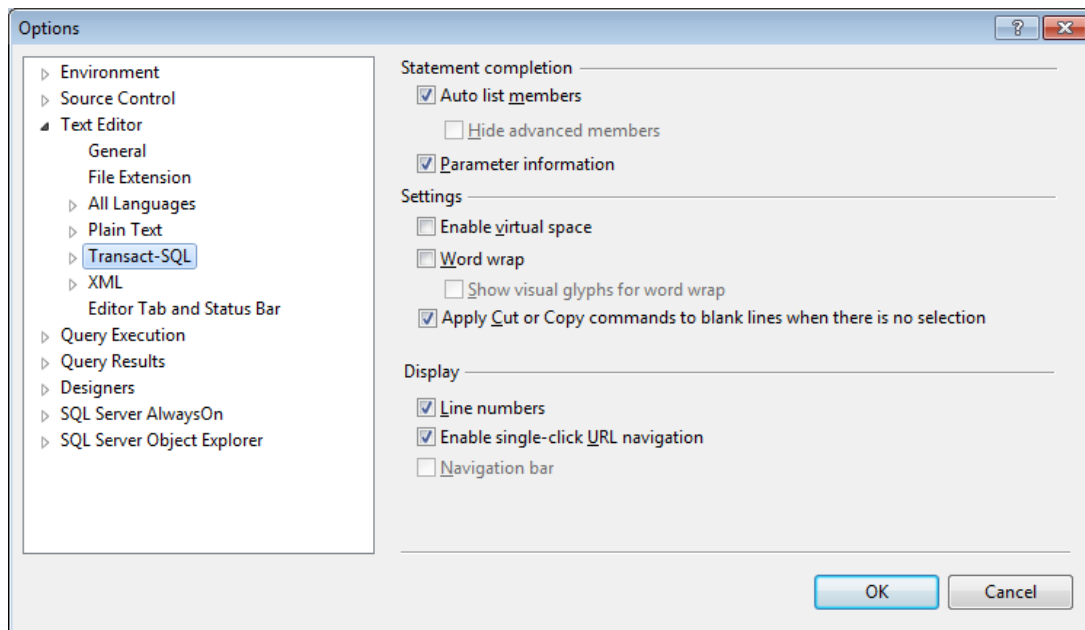
- Selecteer de juist getypte query en druk op **Execute**.

Alleen de geselecteerde query wordt uitgevoerd.
Je krijgt het resultaat zoals hiernaast is weergegeven.

Results		Messages
	naam	plaats
1	Academisch ziekenhuis	Amsterdam
2	Avans Den Bosch	Den Bosch
3	Bosch Medisch Centrum	Den Bosch
4	Fontys Amsterdam	Amsterdam
5	Fontys Eindhoven	Eindhoven
6	Fontys Tilburg	Tilburg

In de schermafbeeldingen in dit lesmateriaal kun je zien dat er regelnummers staan voor de regels in de query-editor. Het is raadzaam om de regelnummering aan te zetten. Bij verwijzingen naar code, kun je dan een regelnummer gebruiken.

- Ga naar de menuoptie **Tools, Options**.
- Kies de groep **Text Editor, Transact-SQL**.
- Vink de optie **Line numbers** aan en druk op **OK**.



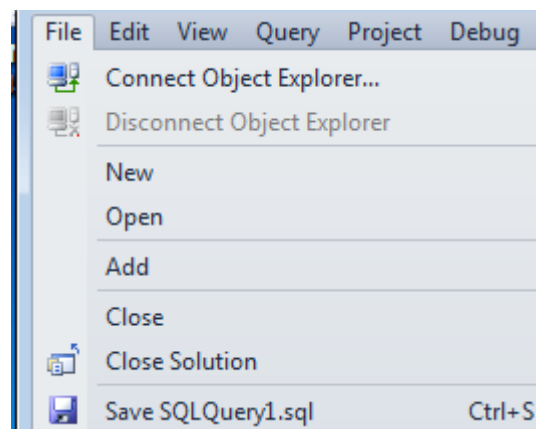
De regelnummering staat voortaan standaard aan.

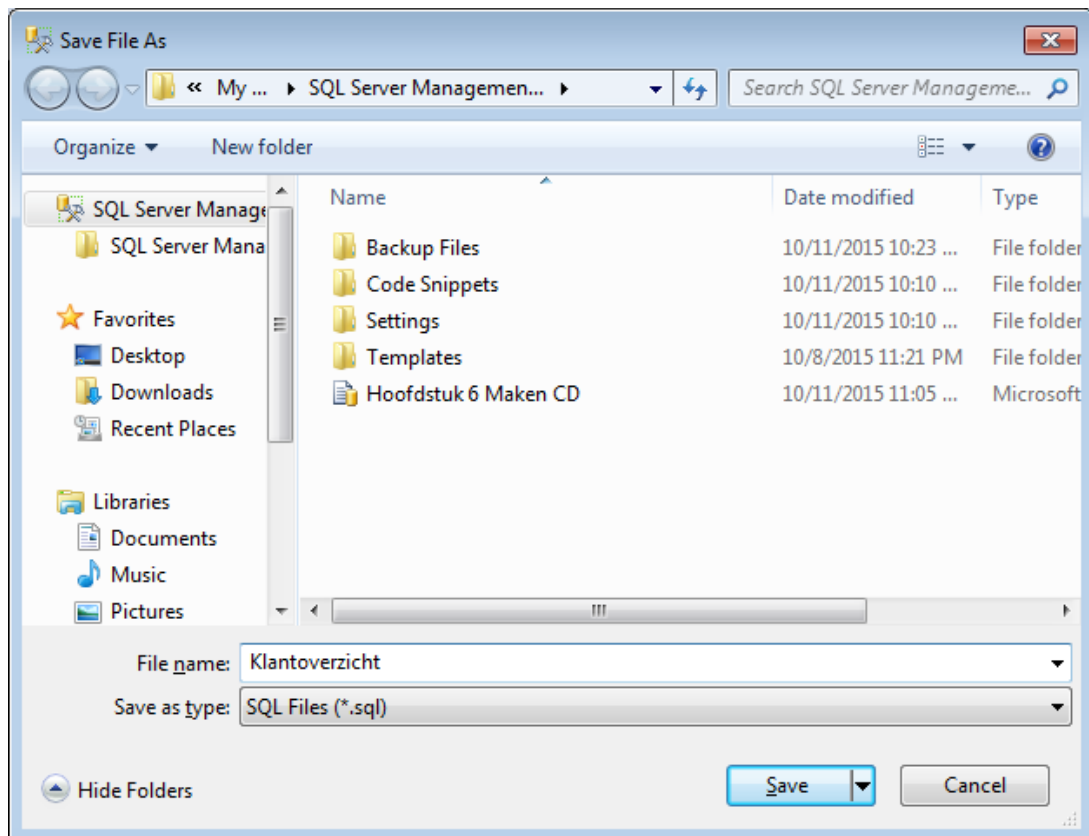
7.5 Het bewaren van een of meer query's

Je hebt in de vorige paragraaf een query gemaakt die de bedrijfsnaam met de plaats laat zien van alle klanten. Je wilt deze query niet iedere keer opnieuw maken. Het is gelukkig mogelijk om een query te bewaren.

- ☞ Ga daarvoor naar de menuoptie **File**.
- ☞ Kies de optie **Save SQLQuery1.sql**

Je krijgt een dialoogvenster te zien waarin je de locatie en de naam van de SQL-query kunt opgeven.





- ☞ Kies altijd een naam waarvan je de functie van de query kunt aflezen. In dit geval noem je de query: **Klantoverzicht**.
- ☞ Bewaar de query door op de knop **Save** te klikken.

7.6 Duplicaten vermijden

In de paragraaf 7.3 heb je een overzicht gemaakt van alle klanten. Je liet van deze gegevens alleen de naam en de woonplaats weergeven. Stel dat je een overzicht wilt maken van alle plaatsen waar klanten zijn gevestigd.

☞ Ga naar het querscherm en vul hierop de onderstaande query in.

select plaats from tblKlant

☞ Voer deze query uit.

Je krijgt het resultaat dat hiernaast is afgebeeld. Het zal je waarschijnlijk wel opvallen dat verschillende woonplaatsen er dubbel in staan.

Hoe kun je dit voorkomen? Om duplicaten in de weergave te voorkomen, gebruik je het statement **SELECT DISTINCT**. Hieronder staat de syntax van het statement weergegeven.

	plaats
1	Amsterdam
2	Den Bosch
3	Den Bosch
4	Amsterdam
5	Eindhoven
6	Tilburg
7	Amsterdam
8	Den Bosch
9	Tilburg

```
SELECT DISTINCT *  
FROM <tabelnaam>
```

of

```
SELECT DISTINCT <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam>
```

Je gaat de query zo aanpassen dat deze elke woonplaats slechts eenmaal laat zien.

☞ Ga naar het querscherm en vul hierop de onderstaande query in.

select distinct plaats from tblKlant

☞ Voer deze query uit.

Je ziet dat elke woonplaats nu slechts eenmaal in het overzicht voorkomt. Dit heb je te danken aan het toevoegen van de optie **DISTINCT**.

	plaats
1	Amsterdam
2	Bodegraven
3	Den Bosch
4	Eindhoven
5	Tilburg

LET OP!

Als je in jouw selectie uit een tabel de primaire sleutel van deze tabel hebt opgenomen, dan zullen er geen duplicaten (kunnen) voorkomen.

Maak nu opdracht 7.1.

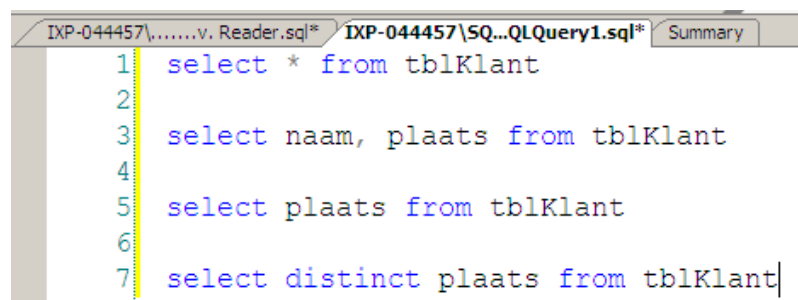
7.7 Commentaar gebruiken?

Je hebt al een aantal query's gemaakt. Op dit moment weet je wat welke query doet en waarvoor deze is gemaakt. De kans is groot dat je echter query's moet gaan maken die je slechts enkele keren per jaar nodig hebt. Het is dan gemakkelijk als je weet wat de query doet. Als iemand anders verder gaat met jouw script, dan is het wenselijk dat deze persoon ook weet waarom bepaalde keuzes zijn gemaakt.

Daarom is het noodzakelijk commentaar in de code toe te voegen bij het maken van query's of scripts. Dit kan ook handig zijn als naslagwerk tijdens toetsen of examens.

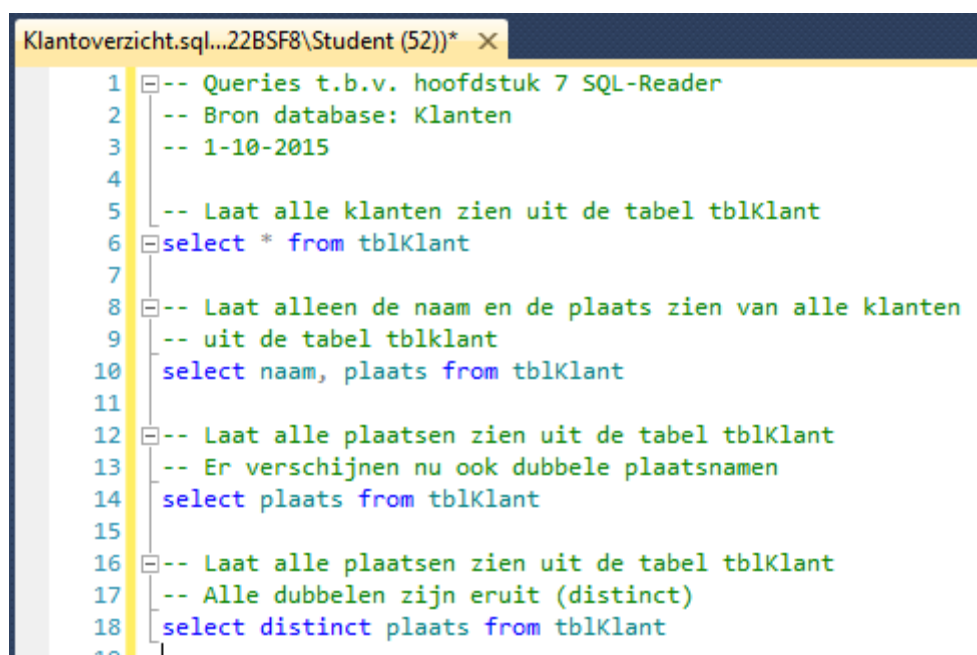
Hieronder zie je twee voorbeelden.

Zonder commentaar



```
IXP-044457\.....v. Reader.sql* IXP-044457\SQL...QLQuery1.sql* Summary
1 select * from tblKlant
2
3 select naam, plaats from tblKlant
4
5 select plaats from tblKlant
6
7 select distinct plaats from tblKlant
```

Met commentaar



```
Klantoverzicht.sql...22BSF8\Student (52))* X
1 -- Queries t.b.v. hoofdstuk 7 SQL-Reader
2 -- Bron database: Klanten
3 -- 1-10-2015
4
5 -- Laat alle klanten zien uit de tabel tblKlant
6 select * from tblKlant
7
8 -- Laat alleen de naam en de plaats zien van alle klanten
9 -- uit de tabel tblKlant
10 select naam, plaats from tblKlant
11
12 -- Laat alle plaatsen zien uit de tabel tblKlant
13 -- Er verschijnen nu ook dubbele plaatsnamen
14 select plaats from tblKlant
15
16 -- Laat alle plaatsen zien uit de tabel tblKlant
17 -- Alle dubbeln zijn eruit (distinct)
18 select distinct plaats from tblKlant
19
```


Alle groene regels in het tweede voorbeeld zijn commentaarregels. In het commentaar omschrijf je kort en bondig wat de query doet en waar de gegevens vandaan komen.

LET OP!

In alle oefeningen en opdrachten die je vanaf nu gaat maken, moet je commentaar verwerken.

7.8 Selecteren met voorwaarden

In de vorige paragrafen heb je gebruikgemaakt van het statement SELECT om bepaalde gegevens uit een tabel zichtbaar te maken. Het nadeel is echter dat je daardoor ALLE klanten te zien krijgt.

In de praktijk zal het vaker voorkomen dat je een bepaalde groep klanten wilt zien. Bijvoorbeeld alle klanten uit Amsterdam of alle klanten die iets gekocht hebben. In die gevallen kan de **WHERE-clausule** aan het statement worden toegevoegd.

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>
FROM <tabelnaam>
WHERE <veldnaam> operator <waarde>
```

Je ziet dat er in het statement het woord **operator** staat. Maar wat is nu eigenlijk een operator? Hieronder sommen we de voornaamste op.

Vergelijkingsoperator	Betekenis
=	Is gelijk aan
<>	Is verschillend van (gebruik alleen bij getallen!)
<	Is kleiner dan
>	Is groter dan
<=	Is kleiner of gelijk aan
>=	Is groter of gelijk aan

Ben je bijvoorbeeld op zoek naar de namen en telefoonnummers van alle klanten uit Amsterdam, dan kun je hiervoor de volgende opdracht gebruiken.

```
SELECT DISTINCT Naam, Telefoonnr
FROM tblKlant
WHERE Plaats = 'Amsterdam'
```

✓ Neem de zojuist gegeven query over.

✓ Voer deze query uit. Je krijgt dan het volgende resultaat.

	Naam	Telefoonnr
1	Academisch ziekenhuis	020-7243239
2	Fontys Amsterdam	020-7463889
3	HAS Amsterdam	020-7243239

Je hebt gezocht naar een veld dat gelijk is aan de tekst die hebt opgegeven. Je gebruikte hiervoor de operator =.

✓ Vraag de namen op van bedrijven die een kredietlimiet hebben van meer dan € 15.000.

✓ Voer de volgende query uit.

```
SELECT Naam, Kredietlimiet
FROM TBLKlant
WHERE Kredietlimiet > 15000
```

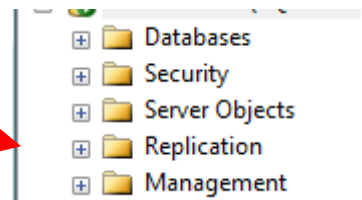
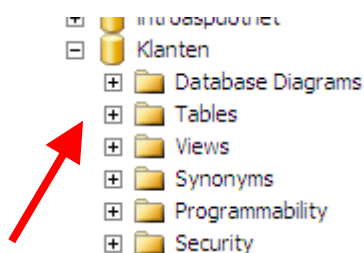
Je krijgt dan het volgende resultaat.

	Naam	kredietlimiet
1	HAS Den Bosch	20000

Je hebt nu twee query's uitgevoerd. Maar hoe kun je nu controleren of het resultaat inderdaad juist is? Dit kun je doen door de inhoud van de betreffende tabel te laten zien.

✓ Ga in de *Object Explorer* van de *Management Studio* naar het onderdeel *Databases*.

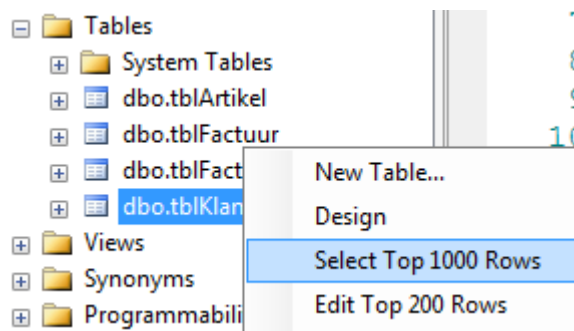
✓ Klik op het plusje voor deze groep om al jouw databases zichtbaar te maken.



✓ Klik op het plusje voor de database Klanten om alle objecten van deze database zichtbaar te maken.

✓ Open de groep *Tables*.

✓ Ga met de muis op de tabel *tblKlant* staan en druk op de rechtermuisknop en kies de optie **Select Top 1000 Rows**.



Je krijgt nu de eerste 1000 records van de tabel te zien.

	klantnr	Naam	Adres	Postcode	Plaats	Directe
1	AMC	Academisch ziekenhuis	Oosterbaan 31	1012XX	Amsterdam	A. Gri
2	AVANS	Avans Den Bosch	Onderwijsboulevard 5	5052DE	Den Bosch	L. Een
3	BMC	Bosch Medisch Centrum	Vijmensseweg 13	5051HT	Den Bosch	C. Ard
4	FontA	Fontys Amsterdam	Den Dam 543a	1018QS	Amsterdam	I. van
5	FontE	Fontys Eindhoven	Philipsstraat 34	5101AZ	Eindhoven	A. Log
6	FontT	Fontys Tilburg	Heuvel 16	5056RR	Tilburg	K. Ker
7	HASA	HAS Amsterdam	Kalverstraat 233b	1009KS	Amsterdam	T. Tor
8	HASD	HAS Den Bosch	Dieze 23	5241KN	Den Bosch	B. Gat
9	KW1	Koning Willem I	Ringbaan Noord 123	5123OP	Tilburg	K. We
10	kw1c	Koning Willem I College	Postbus 38	2410AA	Bodegraven	B. van

Je ziet dat alleen *HAS Den Bosch* een kredietlimiet heeft dat groter is dan € 15.000. Er zijn drie bedrijven met een kredietlimiet van exact € 15.000, maar die krijg je niet te zien met de bovenstaande query.

Let altijd goed op of \geq of $>$ gebruikt moet worden!

Maak nu opdracht 7.2.

7.9 Zoeken op meerdere voorwaarden

In de vorige paragraaf heb je gezocht op de voorwaarde dat de plaatsnaam gelijk is aan 'Amsterdam'. Het zal echter vaker voorkomen dat je iets wilt zoeken dat aan meer voorwaarden tegelijk moet voldoen. In dat geval gebruik je de operatoren **AND** en/of **OR**.

De operator **AND** geeft aan dat aan alle criteria, die door middel van **AND** worden verbonden, moet worden voldaan. De operator **OR** kan worden gebruikt om aan te geven dat moet worden voldaan aan een van de criteria die door middel van **OR** worden verbonden.

De volgende voorbeelden kunnen dit verduidelijken.

☞ Voer de volgende query uit.

```
SELECT Naam, Telefoonnr
FROM tblKlant
WHERE Plaats = 'Amsterdam'
AND Kredietlimiet > 10000
```

Je krijgt nu het onderstaande resultaat te zien.

	Naam	telefoonnr
1	Academisch ziekenhuis	020-7243239

Deze query geeft de namen en telefoonnummers weer van alle klanten uit Amsterdam met een kredietlimiet dat groter is dan € 10.000.

Bij het gebruik van AND moet dus aan beide voorwaarden zijn voldaan.

🖨 Voer nu de onderstaande query uit.

```
SELECT Naam, Telefoonnr
FROM tblKlant
WHERE Plaats = 'Amsterdam'
OR Kredietlimiet > 10000
```

Je krijgt dan het onderstaande resultaat.

	Naam	telefoonnr
1	Academisch ziekenhuis	020-7243239
2	Bosch Medisch Centrum	073-5487124
3	Fontys Amsterdam	020-7463889
4	Fontys Tilburg	020-7243239
5	HAS Amsterdam	020-7243239
6	HAS Den Bosch	020-7243239

Nu worden niet alleen alle klanten uit Amsterdam weergegeven, maar ook alle klanten buiten Amsterdam met een kredietlimiet dat groter is dan € 10.000.

Bij het gebruik van AND krijg je minder records dan bij het gebruik van OR.

7.10 Een bijzonder geval

Je hebt in de vorige paragraaf steeds aangegeven **WAT** je wilt zien. Soms is het echter zinvoller om aan te geven wat je juist **NIET** wilt zien.

In het onderstaande voorbeeld wordt een selectie gemaakt uit de klanten die niet in Amsterdam gevestigd of woonachtig zijn. Je gebruikt daarvoor de operator **NOT**.

Voer de onderstaande query uit.

```
SELECT Naam, Telefoonnr
FROM tblKlant
WHERE NOT Plaats = 'Amsterdam'
```

Je krijgt nu het onderstaande resultaat te zien.

	Naam	telefoonnr
1	Avans Den Bosch	073-6243239
2	Bosch Medisch Centrum	073-5487124
3	Fontys Eindhoven	020-7243239
4	Fontys Tilburg	020-7243239
5	HAS Den Bosch	020-7243239
6	Koning Willem I	013-3740294
7	Koning Willem I College	073-6249909

De operator NOT hoeft echter niet (altijd) te worden gebruikt. Het volgende statement heeft hetzelfde resultaat.

```
SELECT Naam, Telefoonnr
FROM tblKlant
WHERE Plaats <> 'Amsterdam'
```

Toch is het verstandiger om NOT te gebruiken. NOT gaat voor AND of OR. Je kunt dit vergelijken met optellen en vermenigvuldigen. Vermenigvuldigen gaat altijd voor optellen (tenzij je gebruikmaakt van haakjes).

7.11 Wat als een veld leeg is?

Als een veld in een database leeg blijft, zal dit zichtbaar zijn door de tekst **NULL**. Maar hoe moet je nu gaan zoeken op lege velden?

Stel dat je alle klanten wilt weergeven waarbij GEEN kredietlimiet is ingevuld. Je zou verwachten dat je het onderstaande statement kunt gebruiken.

```
SELECT *
FROM tblKlant
WHERE Kredietlimiet = NULL
```

Dit statement geeft geen foutmeldingen, maar zal ook nooit de gewenste gegevens tonen. Je maakt daarom gebruik van het volgende statement.

```
SELECT *  
FROM tblKlant  
WHERE Kredietlimiet IS NULL
```

Maak nu opdracht 7.3 en 7.4.

8 De invoer van gegevens

8.1 Inleiding

Voor het **invoeren van gegevens** kan het DML-statement **INSERT INTO** worden gebruikt. Om dit statement correct te kunnen gebruiken, moet je wel antwoord weten op de onderstaande vragen.

- In welke tabel moeten de gegevens worden ingevoerd?
- Welke velden in die tabel moeten worden gevuld?
- Met welke waarden moeten die velden worden gevuld?

Na bestudering van dit hoofdstuk dient de gebruiker tot het volgende in staat te zijn.

- Het kunnen toevoegen van records aan een tabel.
- Het vullen van een database in de juiste invoervolgorde.

8.2 Syntax

Hieronder is een syntaxoverzicht van de gebruikte statements in dit hoofdstuk.

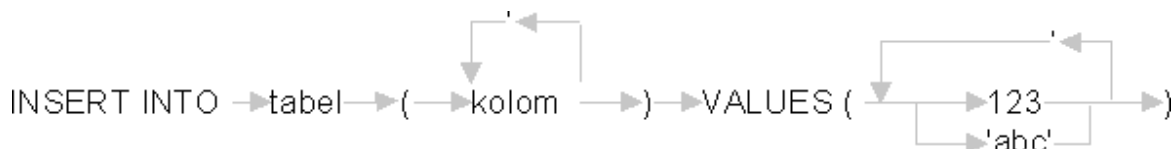
Het toevoegen van een record

INSERT INTO <tabelnaam> (<veld1>, <veld2>, ..., <veldN>)

VALUES (<waarde1>, <waarde2>, ..., <waardeN>)

8.3 INSERT INTO

Hieronder is het syntaxdiagram van het statement INSERT INTO afgebeeld.



Het statement INSERT INTO heeft de volgende syntax.

INSERT INTO <tabelnaam> (<veld1>, <veld2>, ..., <veldN>)
VALUES (<waarde1>, <waarde2>, ..., <waardeN>)

Je gaat een klant toevoegen. Om dit goed te kunnen uitvoeren maak je gebruik van de drie vragen uit de inleiding.

Vraag 1 In welke tabel moeten de gegevens worden ingevoerd?

Alle klanten staan in de tabel *tblKlant*.

Vraag 2 Welke velden moeten worden gevuld?

Om dit te weten moet je eerst de structuur van de tabel kennen. Hieronder staat de structuur afgebeeld.

tblKlant			
	Column Name	Data Type	Allow Nulls
🔑	klantnr	char(5)	<input type="checkbox"/>
	Naam	varchar(30)	<input checked="" type="checkbox"/>
	Adres	varchar(30)	<input checked="" type="checkbox"/>
	Postcode	char(6)	<input checked="" type="checkbox"/>
	Plaats	varchar(20)	<input checked="" type="checkbox"/>
	Directeur	varchar(30)	<input checked="" type="checkbox"/>
	Telefoonnr	char(11)	<input checked="" type="checkbox"/>
	Inschijfnr_kvkk	char(10)	<input checked="" type="checkbox"/>
	Kredietlimiet	int	<input checked="" type="checkbox"/>
	Krediettermijn	int	<input checked="" type="checkbox"/>
	Rentepercentage	int	<input checked="" type="checkbox"/>
	[Klant sinds]	datetime	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Vraag 3 Welke waarden moeten in deze velden staan?

Je voert de onderstaande gegevens in.

Veld	Waarde
Klantnr	Roca
Naam	Roos international
Adres	Pannenschuurlaan 23
Postcode	5061 WE
Plaats	Oisterwijk
Kredietlimiet	10.000
Klant sinds	10-9-2009

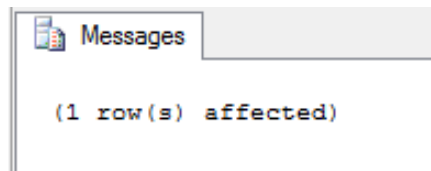
🔗 Neem de onderstaande query over om de klant toe te voegen.

```
INSERT INTO tblKlant (klantnr, Naam, Adres, Postcode,
Plaats, Kredietlimiet, [Klant sinds])
VALUES ('Roca','Roos international','Pannenschuurlaan 23',
'5061WE','Oisterwijk', 10000, '10-9-2009')
```

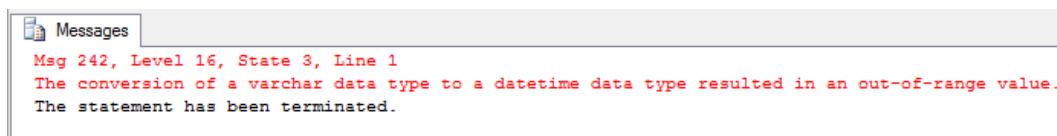

In deze query is duidelijk te zien dat er onderscheid gemaakt wordt tussen de verschillende gegevenstypen. Gegevens van het teksttype (o.a. Varchar, Char, Nvarchar) en data worden tussen enkele aanhalingstekens geplaatst, bij numerieke gegevens gebeurt dat niet. Is de waarde van een veld (nog) niet bekend, dan kan de waarde NULL worden gebruikt. Velden die niet in het statement worden opgenomen, worden automatisch met de waarde NULL gevuld.

SQL staat toe dat veldnamen in tabellen een maximale lengte van 128 karakters hebben, waarin ook spaties mogen worden gebruikt. Worden er inderdaad spaties gebruikt, dan moet in het statement de veldnaam tussen vierkante haken worden geplaatst, zoals al besproken is in hoofdstuk 6 (bijvoorbeeld: [Klant sinds]). In een aantal SQL-varianten is dit echter niet mogelijk. Het beste gebruik je dus geen spaties in tabel- en veldnamen.

Je krijg het onderstaande resultaat te zien.



Als je de onderstaande foutmelding krijgt, is waarschijnlijk het datumformaat niet juist.



De volgorde van de datum is namelijk afhankelijk van de systeeminstellingen van Windows. De volgende varianten zijn mogelijk.

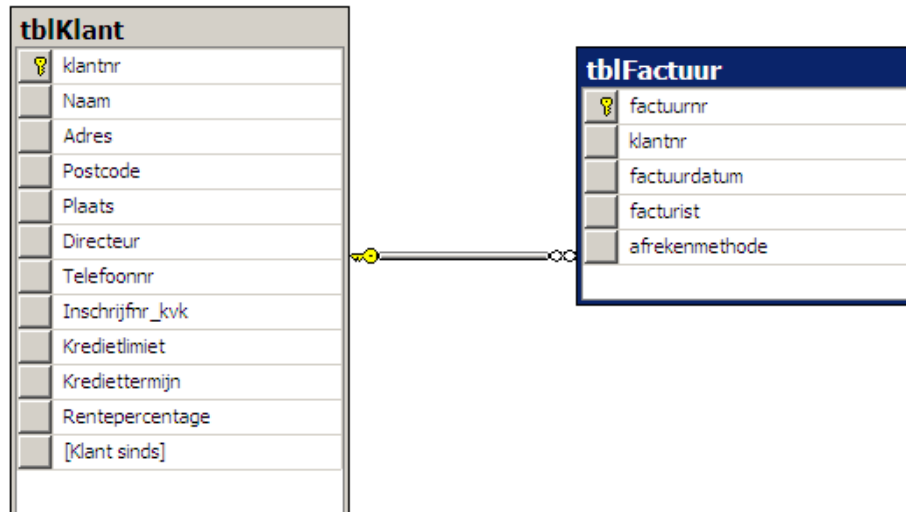
dd-mm-jjjj	10-9-2009
jjjj-mm-dd	2009-09-10 (komt het meest voor)

Je kunt dit eventueel aanpassen door gebruik te maken van het SQL-statement SET DATEFORMAT. Meer informatie hierover kun je terugvinden op de volgende website.

[http://msdn.microsoft.com/en-us/library/ms189491\(SQL.100\).aspx](http://msdn.microsoft.com/en-us/library/ms189491(SQL.100).aspx)

8.4 Invoer van gegevens in meerdere gerelateerde tabellen

Als er gegevens moeten worden ingevoerd in diverse tabellen waartussen een relatie bestaat, dan moet erop worden gelet dat eerst gegevens worden ingevoerd in de tabel aan de 'één-kant' van de relatie als sprake is van een '**één-op-veel relatie**'. Daarna kunnen de gegevens worden ingevoerd in de tabel aan de 'veel-kant' van de relatie.



In de afbeelding hierboven staan de tabellen *tblKlant* en *tblFactuur*. Zoals je kunt zien, hebben deze tabellen een relatie.



Het sleuteltje is de **één-kant** en het achtje op zijn kant (= het wiskundige teken voor oneindig) is de **véél-kant**.

- **klantrnr** is de **foreign key** in de tabel **tblFactuur**.
- **klantrnr** is de **primary key** in de tabel **tblKlant**.

Als je een factuur wilt toevoegen bij een klant die nog niet bestaat, dan moet je eerst de gegevens in het tabel *tblKlant* toevoegen. Pas daarna kun je de factuur toevoegen in de tabel *tblFactuur*. De klant moet dus al bestaan voor je hem een factuur kunt sturen.

Als er in een tabel foreign keys voorkomen, moet je ervoor zorgen dat eerst de tabel waarnaar een foreign key verwijst, gevuld wordt.

Zie paragraaf 3.6 waar uitgelegd staat dat een foreign key altijd verwijst naar de primary key van een andere tabel.

Maak nu opdracht 8.1 + 8.2 + 8.3.

9 Het wijzigen van gegevens

Voor het **wijzigen van gegevens** wordt het DML-statement **UPDATE** gebruikt. Om dit statement correct te kunnen gebruiken moeten enkele zaken bekend zijn.

- In welke tabel moeten de gegevens worden gewijzigd?
- Welke velden in die tabel moeten worden gewijzigd en met welke waarde?
- Aan welk selectiecriterium moet het record voldoen voor de wijziging wordt doorgevoerd?

Na bestudering van dit hoofdstuk dient de gebruiker tot het volgende in staat te zijn.

- Het kunnen wijzigen van gegevens in een tabel.
- Het kunnen werken met *dummy's* om gegevens om te wisselen.
- Het kunnen aanpassen van getallen door middel van een berekening.

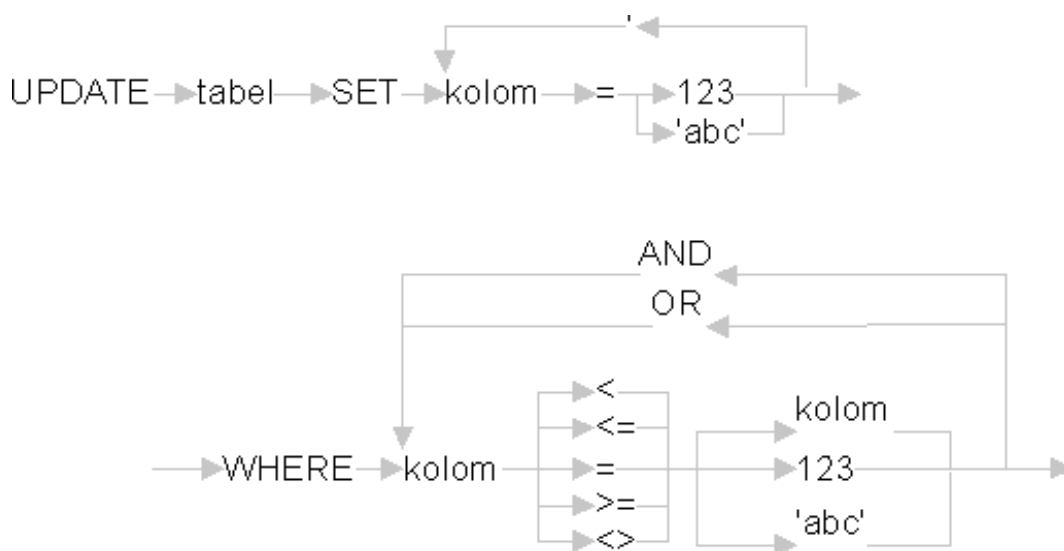
9.1 Syntax

Hieronder is een syntaxoverzicht van de gebruikte statements in dit hoofdstuk.

Het wijzigen van een record	
UPDATE	<tabelnaam>
SET	<veld1> = <waarde1>, <veld2> = <waarde2>
WHERE	<veld> operator <waarde>

9.2 Het statement UPDATE

Hieronder is het syntaxdiagram van het statement UPDATE afgebeeld.



Je gebruikt het statement UPDATE dan ook als volgt.

```
UPDATE <tabelnaam>  
SET <veldnaam1> = <waarde1>  
WHERE <veldnaam2> = <waarde2>
```

9.3 Het aanpassen van een adres

In de tabel *tblKlant* staan alle adresgegevens van de klanten. Bij klant *Roos International* is een fout gemaakt bij het invoeren van het adres.

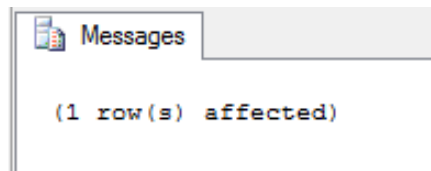
In de database staat *Pannenschuurlaan 23* als adres. Dit moet echter *Zoom 23* zijn.

Je corrigeert deze fout door gebruik te maken van het statement UPDATE.

☞ Neem de onderstaande query over en voer deze uit.

```
UPDATE tblKlant  
SET Adres = 'Zoom 23'  
WHERE Klantnr = 'Roca'
```

Je krijgt het onderstaande resultaat te zien.



☞ Om te controleren of het gelukt is, moet je een select-query uitvoeren. Voer de onderstaande select-query uit.

```
SELECT *  
FROM tblKlant  
WHERE Klantnr = 'Roca'
```

Je zult zien dat de straat inderdaad is aangepast.

NB

Roos International bestaat alleen als de opdrachten in hoofdstuk 8 uitgevoerd zijn.

9.4 Het wijzigen van gegevens uit meerdere gerelateerde tabellen

Het wijzigen van gegevens uit diverse aan elkaar gerelateerde tabellen hoeft op zich geen problemen op te leveren, tenzij de waarde in een sleutelveld moet worden gewijzigd. Zou je proberen het sleutelveld in een tabel aan de 'één-kant' of de vreemde sleutel in een tabel aan de 'veel-kant' te wijzigen, dan betekent dit een inbreuk op de integriteit en kan de wijziging niet worden doorgevoerd. In dat geval moet er gebruikgemaakt worden van een hulprecord.

De klant *Avans Den Bosch* heeft als Klantnr: *AVANS*

Dit gaat problemen opleveren, omdat er verschillende vestigingen van Avans hebben aangegeven klant te worden van ons. Je moet dus het Klantnr gaan aanpassen.

Er zit echter een addertje onder het gras!

Avans Den Bosch heeft als klant al een factuur bij ons. Je kunt dus niet zomaar het klantnr gaan aanpassen. Je zult gebruik moeten gaan maken van een zogenaamde DUMMY.

☞ Als eerste moet je in de tabel *tblFactuur* het klantnr *AVANS* gaan veranderen in *DUMMY*. Hiervoor moet je wel eerst de klant *Dummy* aanmaken.

☞ Voer de onderstaande query uit om de klant *Dummy* aan te maken.

```
INSERT INTO tblKlant (klantnr, naam)
VALUES ('Dummy', 'Dummy klant')
```

☞ Voer dan de volgende query uit om het klantnr aan te passen.

```
UPDATE tblFactuur
SET Klantnr = 'Dummy'
WHERE Klantnr = 'Avans'
```

☞ Daarna zet je het klantnr om van *AVANS* naar *AVAN1*.

```
UPDATE tblKlant
SET Klantnr = 'Avan1'
WHERE Klantnr = 'Avans'
```

☞ In de laatste stap verander je de factuur weer van de dummyklant naar *AVAN1*.

```
UPDATE tblFactuur
SET Klantnr = 'Avan1'
WHERE Klantnr = 'Dummy'
```

Je merkt dat er veel stappen nodig zijn om gegevens aan te passen in databases met relaties. Het statement UPDATE is niet echt lastig, maar door de verschillende relaties in de database, moet je goed nadenken over de stappen die je neemt om iets te kunnen wijzigen.

Om dit soort problemen te omzeilen, worden vaak klantnummers gebruikt in plaats van namen. Als een bedrijf van naam verandert, hoeft het klantnummer niet veranderd te worden. Soms wordt er echter toch een nieuw record aangemaakt voor die klant, om ervoor te zorgen dat in de historie de oude naam en adresgegevens te zien blijven.

LET OP!

Het updaten van records kan niet ongedaan gemaakt worden.

Eventuele problemen zijn te voorkomen door eerst een SELECT-query te maken en deze te testen. Als dan de juiste gegevens verschijnen, kun je van de SELECT-query een UPDATE-query maken.

Het is natuurlijk nog beter om eerst een back-up te maken van de database. Dit komt later in de reader aan bod.

Maak nu opdracht 9.1 + 9.2.

10 Het verwijderen van gegevens

Voor het **verwijderen van gegevens** wordt het DML-statement **DELETE FROM** gebruikt. Om dit statement correct te kunnen gebruiken moeten er enkele zaken bekend zijn.

- Uit welke tabel moeten de gegevens worden verwijderd?
- Aan welk criterium moet het te verwijderen record voldoen?

Na bestudering van dit hoofdstuk dient de gebruiker tot het volgende in staat te zijn.

- Het kunnen verwijderen van gegevens uit een tabel.
- Het kunnen werken met voorwaarden om bepaalde gegevens te verwijderen.

10.1 Syntax

Hieronder is een syntaxoverzicht van de gebruikte statements in dit hoofdstuk.

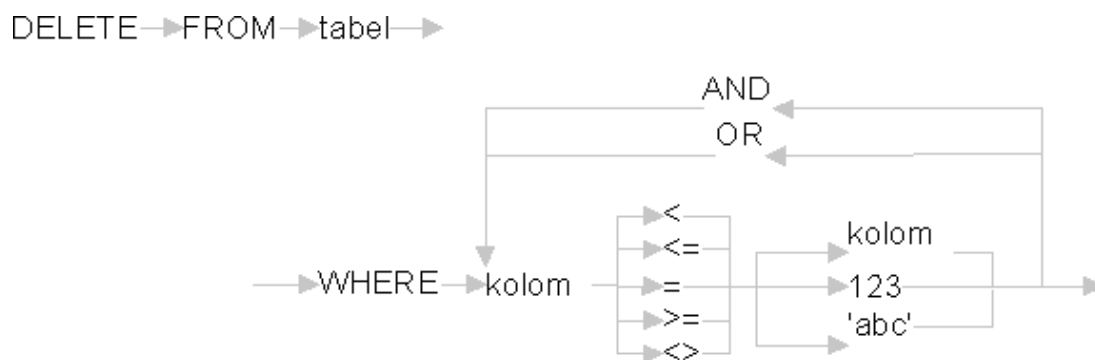
Het wissen van een record

DELETE FROM <tabelnaam>

WHERE <veld> operator <waarde>

10.2 Het statement DELETE

Het syntaxdiagram van het statement DELETE is hieronder weergegeven.



Je moet het statement DELETE dan ook als volgt gebruiken.

DELETE FROM <tabelnaam>
WHERE <veld> operator <waarde>

Op de plaats van de operator kan gebruikgemaakt worden van een van de volgende **vergelijkingsoperatoren**.

Operator	Omschrijving
=	is gelijk aan
>	is groter dan
<	is kleiner dan
<>	is niet gelijk aan
>=	is groter dan of gelijk aan
<=	is kleiner dan of gelijk aan

De klant *Dummy* moet worden verwijderd. Dit doe je als volgt.

Voer de onderstaande query uit.

```
DELETE FROM tblKlant
WHERE Klantnr='Dummy'
```

Je krijgt de melding dat dit is gelukt. Dit werkt echter alleen als je de opdrachten van paragraaf 9.2 hebt uitgevoerd.

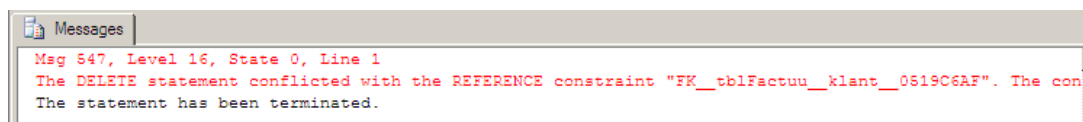
Als alternatief kun je de klant *BMC* verwijderen.

Stel dat je alle klanten met een kredietlimiet van € 10.000 of meer moet verwijderen.

Voer de onderstaande query uit.

```
DELETE FROM tblKlant
WHERE Kredietlimiet >=10000
```

Je krijgt dan de volgende foutmelding te zien.



Dit probleem wordt veroorzaakt door de relaties die bestaan binnen de database. Een aantal klanten heeft facturen. Als je dan de klanten zou kunnen wissen, weet je later nooit meer bij welke klant bepaalde facturen horen. Dit is de reden van het niet kunnen verwijderen van de klant.

10.3 TRUNCATE

Naast het statement DELETE is er nog een andere variant. Deze variant hoef je niet te kennen en zal ook niet in toetsen gevraagd worden.

TRUNCATE

Dit statement is definitief en kan op geen enkele wijze ongedaan gemaakt worden. Hiermee wis je de volledige inhoud van een tabel. Er kan geen keuze worden gemaakt van wat je wilt wissen (WHERE).

Het onderstaande statement wist alle gegevens uit de klanten tabel.

`truncate table tblFactuur`

10.4 Samenvatting

Je hebt nu de vier DML-statements gehad (SELECT, DELETE, UPDATE, INSERT INTO). Het statement SELECT is echter veel uitgebreider dan wat in hoofdstuk 7 aan bod is gekomen. In hoofdstuk 11 en 12 ga je dieper in op het SELECT-statement. De toevoegingen die je in dit hoofdstuk krijgt, moet je ook in de andere DML-statements kunnen toepassen.

Mocht je niet meer precies weten hoe een bepaalde query werkt, zoek het dan nog even na in de reader of gebruik de ingebouwde helpfunctie in SQL (F1).

Maak nu opdracht 10.1.

11 Het selecteren van gegevens deel II

11.1 Inleiding

In hoofdstuk 7 heb je kennisgemaakt met het statement SELECT. In dit hoofdstuk ga je wat dieper in op het statement. Je gaat sorteren/groeperen en berekeningen uitvoeren.

Na bestudering van dit hoofdstuk dient de gebruiker tot het volgende in staat te zijn.

- Het kunnen sorteren en groeperen van gegevens.
- Het kunnen toepassen van ASC en DESC.
- Het kunnen werken met de statistische functies SUM, AVG, MIN en MAX.
- Het kunnen tellen van records door het gebruik van COUNT.

11.2 Syntax

Hieronder staat een syntaxoverzicht van de gebruikte statements in dit hoofdstuk.

Het sorteren van de selectie

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam>  
ORDER BY <veld1>, <veld2>, ..., <veldN> ASC | DESC
```

Het groeperen van een selectie

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam>  
GROUP BY <veld1>, <veld2>, ..., <veldN>
```

Het beperken van de groepering

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam>  
GROUP BY <veld1>, <veld2>, ..., <veldN>  
HAVING COUNT(<veld>) operator <aantal>
```

11.3 De uitvoer sorteren

De meeste gegevens die je uit een database haalt door gebruik te maken van SELECT moeten op de een of andere manier gesorteerd worden. Dit kun je doen met behulp van ORDER BY.

Achter **ORDER BY** komen een of meer sorteervelden, eventueel gevolgd door **ASC** of **DESC**.

ASCending betekent *oplopend*: van klein naar groot bij getallen, van A naar Z bij tekst.

DESCending betekent *aflopend*: van groot naar klein en van Z naar A.

Een tweede of volgend sorteerveld wordt alleen gebruikt wanneer de inhoud van de vorige sorteervelden gelijk is. De algemene syntax luidt als volgt.

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>
FROM <tabelnaam>
ORDER BY <veld1>, <veld2>, ..., <veldN> ASC | DESC
```

Stel dat je een overzicht op alfabetische volgorde wilt hebben van alle plaatsen waarin bedrijven zijn gevestigd.

☞ Neem de onderstaande query over en test deze uit.

```
SELECT DISTINCT Plaats
FROM tblKlant
ORDER BY Plaats
```

Je krijgt het resultaat zoals hiernaast is weergegeven.

Als je geen ASC of DESC gebruikt, zal er altijd gesorteerd worden van A tot Z.

	Plaats
1	Amsterdam
2	Bodegraven
3	Den Bosch
4	Eindhoven
5	Tilburg

☞ Je past de query nu zo aan dat de plaatsen gesorteerd worden van Z tot A.

```
SELECT DISTINCT Plaats
FROM tblKlant
ORDER BY Plaats DESC
```

Het resultaat is dan inderdaad in omgekeerde volgorde gesorteerd.

Stel: er moet een uitgebreide mailing plaatsvinden naar alle klanten. Het verzendbedrijf biedt een korting in de verzendkosten als de enveloppen gesorteerd op postcode en adres worden aangeleverd.

☞ Voer de onderstaande query uit.

```
SELECT Naam, Adres, Postcode, Plaats
FROM tblKlant
ORDER BY plaats DESC, Postcode ASC
```

Je ziet dat er aflopend (DESC) wordt gesorteerd op plaats en bij de records met dezelfde plaats wordt oplopend (ASC) gesorteerd op postcode.

☞ Je moet dan ook het volgende resultaat krijgen.

	Naam	Adres	Postcode	Plaats
1	Fontys Tilburg	Heuvel 16	5056RR	Tilburg
2	Koning Willem I	Ringbaan Noord 123	5123OP	Tilburg
3	Fontys Eindhoven	Philipsstraat 34	5101AZ	Eindhoven
4	Bosch Medisch Centrum	Vijmenseweg 13	5051HT	Den Bosch
5	Avans Den Bosch	Onderwijsboulevard 5	5052DE	Den Bosch
6	HAS Den Bosch	Dieze 23	5241KN	Den Bosch
7	Koning Willem I College	Postbus 38	2410AA	Bodegraven
8	HAS Amsterdam	Kalverstraat 233b	1009KS	Amsterdam
9	Academisch ziekenhuis	Oosterbaan 31	1012XX	Amsterdam
10	Fontys Amsterdam	Den Dam 543a	1018QS	Amsterdam

11.4 De uitvoer groeperen

Soms moet je in bepaalde gevallen gegevens niet alleen sorteren maar ook groeperen. Je wilt bijvoorbeeld het aantal klanten per woon- of vestigingsplaats weergeven.

Je gebruikt hiervoor de clausule **GROUP BY**.

De algemene syntax luidt dan als volgt.


```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>
FROM <tabelnaam>
GROUP BY <veld1>, <veld2>, ..., <veldN>
```

Je moet echter de clausule GROUP BY altijd gebruiken met een rekenkundige functie (bijvoorbeeld optellen, gemiddelde, laagste).

Stel dat je wilt weten hoeveel klanten je per plaats hebt. Dan wil je dus het aantal records tellen. Je gebruikt dan een ingebouwde functie waarmee aantallen kunnen worden bepaald: **COUNT**.

Het voorbeeld kan dan als volgt worden uitgewerkt.

```
SELECT Plaats, COUNT(Naam)
FROM tblKlant
GROUP BY Plaats
```

 Voer de bovenstaande query uit. Je krijgt dan het volgende resultaat.

	Plaats	(No column name)
1	Amsterdam	3
2	Bodegraven	1
3	Den Bosch	3
4	Eindhoven	1
5	Tilburg	2

Je ziet dat eerst de plaatsnamen zijn weergegeven en daarnaast het aantal klanten. Deze kolom heeft echter geen naam ('No column name').

Dat wil je natuurlijk niet op je overzicht hebben. Je past de query daarom zo aan dat deze kolom wel een goede titel krijgt.

☞ Pas de query aan zoals hieronder is aangegeven en voer deze uit.

```
SELECT Plaats, COUNT(Naam) as Aantal
FROM tblKlant
GROUP BY Plaats
```

	Plaats	Aantal
1	Amsterdam	3
2	Bodegraven	1
3	Den Bosch	3
4	Eindhoven	1
5	Tilburg	2

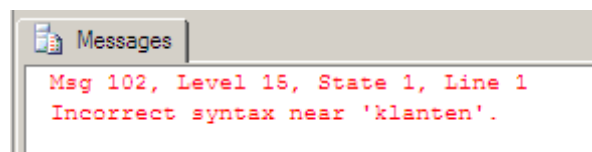
Je ziet dat je een overzicht krijgt met de gegeven kolomnaam *Aantal*.

☞ Stel dat je als kolomnaam *Aantal klanten* wilt gebruiken. Dan kom je een probleem tegen.

☞ Pas de query aan zoals hieronder is weergegeven.

```
SELECT Plaats, COUNT(Naam) as Aantal klanten
FROM tblKlant
GROUP BY Plaats
```

Als je de query nu gaat uitvoeren krijg je de onderstaande foutmelding.



Het probleem is dat je GEEN spaties mag gebruiken in namen. Maar hoe kan je er dan voor zorgen dat je toch de titel *Aantal klanten* kunt gebruiken? Dit doe je door gebruik te maken

van vierkante haakjes []. Alle tekst die je tussen deze haakjes plaatst, wordt behandeld als één veld of item.

☞ Pas de query aan zoals hieronder is weergegeven en bekijk het resultaat.

```
SELECT Plaats, COUNT(Naam) as [Aantal klanten]
FROM tblKlant
GROUP BY Plaats
```

Achter COUNT kun je tussen haakjes elke willekeurige (maar bestaande) veldnaam opnemen. Het gaat erom dat er een veld kan worden gebruikt om de telling daadwerkelijk te kunnen uitvoeren. In plaats van een veldnaam kan ook de asterisk (*) worden toegepast.

De naam die gegeven is aan een kolom (zoals *Aantal klanten*) kan ook gebruikt worden achter het sleutelwoord FROM. Je kunt bijvoorbeeld sorteren op [Aantal klanten].

11.5 De groepering beperken

In de voorgaande paragraaf wordt door middel van een query met de clause GROUP BY het aantal klanten per plaats bepaald.

Stel dat dit aantal alleen moet worden weergegeven als dat groter is dan 2. In dat geval kan gebruik worden gemaakt van de clause **HAVING**. De query kan dan als volgt worden samengesteld.

```
SELECT Plaats, COUNT(*)
FROM tblKlant GROUP BY Plaats
HAVING COUNT(*) > 2
```

☞ Neem de bovenstaande query over en voer deze uit. Je krijgt dan het onderstaande resultaat.

	Plaats	(No column name)
1	Amsterdam	3
2	Den Bosch	3

- ☞ Je ziet dat je alleen de plaatsen te zien krijgt met meer dan twee klanten. Als je alle plaatsen wilt zien met twee of meer klanten, dan moet je de query als volgt aanpassen.

```
SELECT Plaats, COUNT(*)
FROM tblKlant
GROUP BY Plaats
HAVING COUNT(*) >= 2
```

- ☞ Voer de bovenstaande query uit en je zult zien dat ook de plaats Tilburg wordt weergegeven.

11.6 Andere ingebouwde functies

Naast COUNT dat je in de vorige paragraaf hebt gebruikt, zijn er nog vier andere functies die je moet kennen.

Functie	Omschrijving
SUM	voor het berekenen van de som
AVG	voor het berekenen van het gemiddelde
MIN	voor het bepalen van de kleinste waarde
MAX	voor het bepalen van de grootste waarde

In de hierna volgende voorbeelden zie je hoe deze functies gebruikt worden.

Bepaal de totale kredietlimiet van alle klanten samen.

```
SELECT SUM(Kredietlimiet)
FROM tblKlant
```

Bepaal de gemiddelde kredietlimiet.

```
SELECT AVG(Kredietlimiet)
FROM tblKlant
```

Wat is de laagste kredietlimiet?

```
SELECT MIN(Kredietlimiet)
FROM tblKlant
```

Wat is de hoogste kredietlimiet?

```
SELECT MAX(Kredietlimiet)
FROM tblKlant
```



Test deze query's uit.

Maak nu opdracht 11.1.

12 Meer dan een tabel gebruiken en subquery's

12.1 Inleiding

In de vorige hoofdstukken heb je telkens query's uitgevoerd op één tabel. In de praktijk is het raadplegen van gegevens over het algemeen niet beperkt tot één tabel. In dit hoofdstuk ga je kijken hoe je gegevens uit diverse tabellen kunt raadplegen. Tevens ga je werken met subquery's.

Na bestudering van dit hoofdstuk dient de gebruiker tot het volgende in staat te zijn.

- Het selecteren van verschillende velden uit meer dan één tabel.
- Een relatie kunnen aangeven tussen twee of meer tabellen.
- Het kunnen maken en uitvoeren van een subquery.

12.2 Syntax

Hieronder is een syntaxoverzicht van de gebruikte statements in dit hoofdstuk.

Het selecteren van gegevens uit twee tabellen met een INNER JOIN

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam1>, <tabelnaam2>  
WHERE <tabelnaam1>.<relveld1> = <tabelnaam2>.<relveld2>
```

Het selecteren van gegevens uit een ander selectie van gegevens door gebruik te maken van een subquery

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam>  
WHERE <veld> operator  
(SELECT (DISTINCT) <veld1>, ..., <veldN>  
FROM <tabelnaam>  
WHERE <veld> operator <waarde>)
```

Een selectie door gebruik te maken van de IN-operator

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam>  
WHERE <veld> IN (<waarde1>, ..., <waardeN>)
```

Het gebruik van pseudoniemen

```
SELECT (DISTINCT) <veld1>, <veld2>, ..., <veldN>  
FROM <tabelnaam> AS <pseudoniem>
```

12.3 Gegevens uit meer dan een tabel

Stel dat je de uitgereikte facturen per klant wilt raadplegen. In principe kun je dan volstaan met het raadplegen van de tabel *TBL_Factuur*, omdat je natuurlijk alle facturen kunt sorteren op klantnummer en vervolgens op factuurnummer. Alle records in de tabel *TBL_Factuur* worden dan in de eerste plaats op klantnummer gesorteerd en vervolgens per klant op factuurnummer.

Een klantnummer alleen zegt echter niet zoveel. Het overzicht krijgt meer betekenis als je ook de naam van de klant, het adres, de postcode en de woonplaats in het overzicht kunt opnemen. In dat geval moet je gebruikmaken van de gegevens in twee tabellen: *TBL_Factuur* en *TBL_Klant*.

Voor het raadplegen van gegevens uit meer dan één tabel heb je twee mogelijkheden.

- Het gebruik van een **subquery**, waarbij meerdere SELECT-opdrachten in één opdracht worden toegepast, en
- het gebruik van een **JOIN-query**, waarbij in het FROM-gedeelte van de SELECT-opdracht meer dan één tabel wordt opgenomen.

12.4 De JOIN-query

Zoals je hierboven hebt kunnen lezen, wordt de **JOIN-query** gebruikt om gegevens uit meer dan een tabel te raadplegen. In dergelijke gevallen kun je ook (vaak) de subquery toepassen, maar in een **RDBMS** leiden JOIN-query's tot betere prestaties.

In het begin van de vorige paragraaf werd geopperd dat je misschien geïnteresseerd was in een overzicht van de facturen per klant (eventueel gesorteerd op klantnummer en vervolgens op factuurnummer). In een JOIN-query worden de betrokken tabellen opgenomen in het FROM-gedeelte van de query.

```
SELECT TBL_klanten.klantnummer, klantnaam, adres, postcode, woonplaats, factuurnummer,  
factuurdatum  
FROM TBL_klanten, TBL_Factuur  
WHERE TBL_klanten.klantnummer = TBL_Factuur.klantnummer
```

Het veld *klantnummer* komt in beide tabellen voor. Je moet er dus voor zorgen dat je een duidelijk onderscheid tussen beide velden maakt om foutmeldingen te voorkomen. Je doet dit door de veldnaam te laten voorafgaan door de tabelnaam en er een punt tussen te zetten. De andere veldnamen komen slechts in een van beide tabellen voor.

12.5 Het Cartesiaanse product

Als je probeert gegevens te retourneren uit twee of meer tabellen zonder JOINS te gebruiken, dan maak je een zogenaamd 'Cartesiaans product'.

Een Cartesiaans product kun je het beste als volgt omschrijven.

Alle mogelijke combinaties van rijen in alle tabellen.

LET OP!

Zorg ervoor dat je JOINS hebt, voordat je probeert gegevens te retourneren. Het kan uren duren, voordat een Cartesiaans product voor tabellen met veel records en/of voor veel tabellen is voltooid.

Als je drie tabellen hebt met respectievelijk 100, 1000 en 10.000 records, dan is het resultaat van een query zonder JOINS: $100 \times 1000 \times 10.000 = 1.000.000.000$ records.

12.6 De subquery

In een subquery worden verschillende SELECT-opdrachten samengevoegd in een opdracht. Bij een subquery worden over het algemeen geen diverse tabellen geraadpleegd, maar worden tabellen meermaals geraadpleegd. Anders gezegd: het resultaat van een query wordt nogmaals geraadpleegd, of het resultaat wordt gebruikt om een nieuwe query uit te voeren.

Je wilt bijvoorbeeld een overzicht van alle klanten die in dezelfde plaats gevestigd zijn als de klant 'Instruct'. Je kunt dan eerst een query toepassen om de vestigingsplaats van de klant te vinden en vervolgens dit resultaat in een nieuwe query gebruiken.

```
SELECT woonplaats  
FROM TBL_klanten  
WHERE naam = 'Instruct'
```

Het resultaat zal 'Bodegraven' zijn en dit neem je vervolgens op in de query.

```
SELECT naam  
FROM TBL_klanten  
WHERE woonplaats = 'Bodegraven'
```

In een subquery kunnen deze twee query's worden samengevoegd tot een (samengestelde) query.

```
SELECT naam
FROM TBL_klanten
WHERE woonplaats =
(SELECT woonplaats
FROM TBL_klanten
WHERE naam = 'Instruct')
```


Nu zit hier echter wel een addertje onder het gras. In de hiervoor weergegeven subquery mag de subvraag niet meer dan één resultaat opleveren. Komt er in het klantenbestand nog een klant met de naam 'Instruct' voor in een andere vestigingsplaats dan Bodegraven, dan levert dit een foutmelding op.

Je kunt de foutmelding voorkomen door gebruik te maken van de operator **IN**.

```
SELECT naam
FROM TBL_klanten
WHERE woonplaats IN
(SELECT woonplaats
FROM TBL_klanten
WHERE naam = 'Instruct')
```

12.7 De IN-operator

Condities kunnen wel eens lastig worden als je wilt weten of de waarde van een kolom voorkomt in een gegeven verzameling waarden waarbij die verzameling ook nog eens groot is. Hieronder staat er een voorbeeld van.

-  Geef alle namen van klanten die gevestigd zijn in Tilburg, Eindhoven, Breda of Den Bosch.

```
SELECT Naam
FROM tblKlanten
WHERE plaats = 'Tilburg'
OR plaats = 'Eindhoven'
OR plaats = 'Breda'
OR plaats = 'Den Bosch'
```

De instructie is goed, alleen wat lang. Met de IN-operator kan de instructie vereenvoudigd worden.

```
SELECT Naam
FROM tblKlanten
WHERE plaats IN ('Tilburg', 'Eindhoven', 'Breda', 'Den Bosch')
```

Voor de expressies achter de IN-operator gelden de volgende regels.

- De datatypes van de expressies moeten vergelijkbaar zijn.
- Statistische functies (MIN, MAX, COUNT, SUM, AVG) zijn als expressies niet toegestaan.

12.8 Het pseudoniem

Indien diverse tabelspecificaties in de FROM-component voorkomen, is het soms gemakkelijk zogenaamde *pseudoniemen* te gebruiken. Een andere naam die wel eens wordt gebruikt voor *pseudoniem* is *alias*.

Pseudoniemen zijn tijdelijke alternatieve namen voor tabelnamen. In de vorige voorbeelden herhaalden we de tabelnamen steeds in hun geheel. We kunnen in plaats van de tabelnamen ook pseudoniemen gebruiken, zoals in dit voorbeeld.

```
SELECT K.Klantnr, Naam, Plaats, Factuurnr
FROM tblKlant K, tblFactuur F
WHERE K.Klantnr = F.Klantnr
```

of

```
SELECT K.Klantnr, Naam, Plaats, Factuurnr
FROM tblKlant AS K, tblFactuur AS F
WHERE K.Klantnr = F.Klantnr
```

In de FROM-component staan achter de tabelnamen de pseudoniemen vermeld ofwel *gedeclareerd*. In de andere componenten kunnen we deze pseudoniemen gebruiken in plaats van de werkelijke tabelnamen. Het feit dat in de instructie het pseudoniem 'K' eerder gebruikt wordt dan dat het gedeclareerd wordt, levert geen problemen op. De FROM-component wordt namelijk als eerste verwerkt.

Maak nu opdracht 12.1 + 12.2.