Relationele databases en SQL

Bij deze lesstof hoort de Access-database ‘Bibliotheek.mdb’

1 STRUCTUUR IN INFORMATIE: NAAR EEN RELATIONELE DATABASE

Een gegevensbank is een verzameling van gegevens. Gegevensbanken zijn er in allerlei soorten en maten, denk aan een telefoonklapper of een adresboekje. In een telefoonklapper noteer je de telefoonnummers van vrienden en bekenden. De structuur van de gegevensbank is eenvoudig. Er is ruimte om namen en telefoonnummers op te schrijven. De pagina's zijn geordend op de beginletter van de naam om het opzoeken van telefoonnummers eenvoudig te maken.
Wanneer je de computer gebruikt, heb je kleine gegevensbanken in allerlei programma's. Denk bijvoorbeeld aan een adressenboekje bij je e-mailprogramma. Dit zijn meestal lijstjes van adressen die simpel te gebruiken zijn.
De zaak verandert als de gegevensbank groter wordt. Als voorbeeld bekijken we de schooladministratie. Nog niet zo lang geleden werd deze volledig met kaartenbakken gedaan. Op sommige scholen vind je misschien nog steeds zulke bakken, ergens in de kelder of zo. De administratie hield van de leerlingen kaarten bij. Daarop stonden naam, adres, woonplaats, telefoonnummer en waarschijnlijk nog wat persoonlijke details. In de kaartenbak werden speciale voorgedrukte kaarten gebruikt. Deze kaarten hadden invulruimtes, velden. In deze velden konden de gegevens worden opgeschreven of met de typemachine ingevuld. De kaarten hadden nog open ruimtes, zodat later nog gegevens konden worden toegevoegd.
De schooladministratie stopt natuurlijk niet bij het bijhouden van gegevens over leerlingen. Er moeten ook gegevens over docenten, klassen, roosters enzovoort worden beheerd. Van de leerlingen moeten ook de studieresultaten worden bijgehouden. Dit gebeurde in andere kaartenbakken of met behulp van speciale middelen. Denk bijvoorbeeld aan een groot planbord voor het lesrooster. Een geautomatiseerd systeem op de computer kan deze kaartenbakken vervangen. We spreken pas van een database als we het hebben over een geautomatiseerde
! gegevensbank, waarbij de gegevens onderhouden worden door een databasemanagementsysteem. Een telefoonklapper is dus geen database!
Er bestaat standaardsoftware om gegevensbanken te maken en te beheren. We noemen dit een DataBase Management System (DBMS, in het Nederlands Gegevensbankbeheersysteem). Een database zonder DBMS is als een Word-document zonder het programma Word: je kunt er niets mee. Bekende DBMS'en zijn Oracle, FileMaker, Microsoft Access en Firebird.
De eerste belangrijke taak van een databasesysteem is het opslaan van gegevens en het mogelijk maken van wijzigingen op deze gegevens. Dit zal niet de enige taak van het DBMS zijn. De gegevens worden gebruikt om allerlei taken goed uit te kunnen voeren. Mensen moeten informatie kunnen opvragen of afleiden uit de opgeslagen gegevens. Ook kunnen er allerlei bewerkingen op de gegevens worden uitgevoerd; denk aan het maken van klassenlijsten, adresetiketten, berekening van (eind)cijfers enzovoort.
Een tweede taak van een databasesysteem is het bieden van mogelijkheden voor het opvragen en verwerken van informatie.
Er is nog een derde taak die te maken heeft met het voorkomen van fouten in de database. In elke administratie worden fouten gemaakt. We bekijken er een paar.

- Het zou kunnen voorkomen dat een leerling per ongeluk twee kaarten heeft gekregen. Zo'n probleem kan lang onopgemerkt blijven, tot er bijvoorbeeld een adreswijziging binnenkomt die maar op één van de twee kaarten wordt doorgevoerd.
- Als kaarten onvolledig zijn ingevuld, kunnen er ook problemen ontstaan omdat belangrijke informatie ontbreekt. De kaartenbak kan niet afdwingen dat iemand

ook echt iets invult,je kunt hoog uit op de voorgedrukte kaart een opmerking zetten: Vul dit altijd in.
- Je moet ervoor zorgen dat de leerlingen uit elkaar worden gehouden. Naamsverwarring is een groot probleem in administraties: er zullen ongetwijfeld leerlingen met dezelfde voor- en achternaam op school zijn. Maar ook broers en zussen hebben veel dezelfde gegevens.
- Met betrekking tot de verschillende onderdelen van de administratie kunnen fouten optreden. Op het roosterplanbord kun je bijvoorbeeld per ongeluk twee klassen in hetzelfde lokaal plannen.

In een papieren administratie hangt het correct houden van de gegevens helemaal af van de gebruiker. Papier is passief, het kan niet roepen als er iets verkeerds wordt ingevuld. Een databasesysteem is een programma dat kan reageren op de gegevens die door een gebruiker worden ingevoerd. Hierdoor kan een deel *van* de mogelijke fouten worden voorkomen. Zo zal een databasesysteem een foutmelding kunnen geven als er een klas op het rooster wordt gezet die verder nergens in de database voorkomt. Natuurlijk kan een databasesysteem niet alles controleren, een verkeerd gespelde naam leidt niet tot een foutmelding. Het systeem kan wel helpen bij het correct houden van de gegevens.
We noemen dit het bewaken van de integriteit van de database. Hiermee bedoelen we dat het systeem kan controleren of de ingevoerde gegevens aan een aantal regels voldoen.

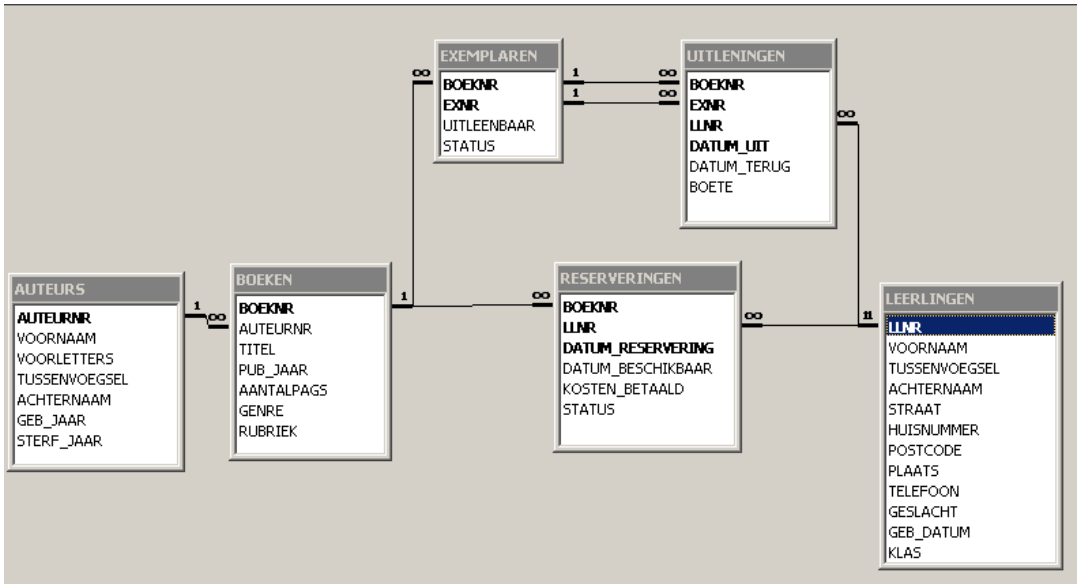
Het relationele model
Wanneer databases wat groter worden, is het erg belangrijk dat ze goed georganiseerd zijn. Dit is nodig om goed beheer mogelijk te maken en ervoor te zorgen dat informatie uit de database opgevraagd kan worden. Een goede manier om een database te organiseren is het relationele model. Wanneer een gegevensbank georganiseerd wordt volgens het relationele model, komen alle gegevens in tabellen. DBMS'en die werken volgens het relationele model noemen we Relational DataBase Management Systems (RDBMS)
! Gegevens van één leerlingkaart in de kaartenbak komen nu op één rij in de tabel. De velden op de systeemkaart worden de kolommen van de tabel.
Bij het relationele model hoort een taal waarmee je de database benadert. Meestal is dat om informatie uit de database te halen, maar ook wel om de database bij te werken en het gebruik te regelen. Zoals het in de automatiseringswereld heel gebruikelijk is, kent deze programmeertaal alleen Engelse woorden.
Het Engelse woord voor een informatievraag is query. De vraagtaal heet daarom SQL, Structured Query Language, oftewel Gestructureerde Vraagtaal. Bij het relationele model hoort ook een taal om de structuur van een database vast te leggen. Daarmee vertel je het systeem welke tabellen er zijn en welke kolommen die tabellen hebben. Je kunt dan ook beperkingen invoeren, waarmee het systeem de integriteit van de database kan bewaken. Zo kun je opgeven dat bepaalde gegevens altijd ingevuld moeten worden of dat er bij 'geslacht' alleen 'm' of 'v' kan worden ingevoerd. Dit helpt om de eerder genoemde fouten te voorkomen. Er bestaan veel verschillende DBMS'en voor het werken met relationele databases. De grotere, professionele systemen kunnen omgaan met SQL. Daarnaast hebben ze aanvullende middelen, bijvoorbeeld om een database te definiëren en te vullen. Dit gaat in SQL namelijk niet zo soepel. Ze bevatten ook vele extra's, zoals middelen om invulformulieren te maken en rapportages af te drukken.
In deze lesbrief beperken we ons tot de standaardonderdelen die in alle relationele databaseprogramma's zitten. Het relationele model is bedacht door de wiskundige E.F. Codd († 2 mei 2003), toen werkzaam bij computergigant IBM, en gepubliceerd in 1970. De term relationeel is afkomstig uit de wiskunde. In de verzamelingenleer bestaat het begrip relatie. Het idee is dat je met zinnen als 'Leerling x zit in klas y'

een relatie legt tussen leerlingen en klassen. Bijvoorbeeld 'Jantine de Bakker zit in 4b'. De elementen van deze relatie geven de kolommen leerling en klas. In een rij van de tabel komen dan 'Jantine de Bakker' en '4b'.

Tabel: Klasindeling	
Leerling	Klas
Jantine Bakker	4b

2 VOORBEELD VAN EEN DATABASE

In dit hoofdstuk gebruiken we een grote database om te leren werken met de vraagtaal SQL. Het is een database die informatie bevat over een schoolbibliotheek. Daar zitten gegevens in over de boeken en over de leerlingen die ze lenen.
De database heeft zes tabellen. Elke tabel heeft weer enkele kolommen. De kolomnamen zijn zo gekozen dat ze zichzelf verklaren. Soms krijg je erg lange uitdrukkingen die je weer wilt inkorten. Bijvoorbeeld Geb_datum in plaats van Geboortedatum.
Een boek - een werk dat geschreven is door een schrijver - is iets anders dan een exemplaar van een uitgave van dat boek. Van boeken waar veel vraag naar is, zullen er verschillende exemplaren in de bibliotheek zijn. Daarom zijn er twee tabellen: Boeken en Exemplaren. De exemplaren worden aangeduid met het boeknummer (Boeknr) en een exemplaarnummer (Exnr).Je kunt een boek reserveren, maar je leent een exemplaar, het eerste dat beschikbaar komt.
Bij de tabel Reserveringen wordt bijgehouden wie welk boek wanneer reserveert, wanneer het beschikbaar is, of het al is opgehaald en welke kosten eraan verbonden zijn. Als reserveren gratis zou zijn, dan zou er wellicht meer gereserveerd dan gelezen worden. Sommige boeken worden niet uitgeleend. Bijvoorbeeld naslagwerken. Dat wordt in de tabel Exemplaren in de kolom Uitleenbaar bijgehouden. Of een exemplaar uitgeleend, beschikbaar, in reparatie of gewoon zoek is, wordt bijgehouden in de kolom Status.
De structuur van de database met de tabellen en kolommen wordt vaak weergegeven met een zgn strokendiagram. In Access kun je de tabellen en de verbanden zichtbaar maken met de optie Relaties in het menu Extra:



Je ziet dat er kolommen in elke tabel voorkomen die alleen maar een nummer bevatten. Leerlingnr, Boeknr, Exnr en Auteurnr. Dat lijken overbodige kolommen, maar er zijn heel goede redenen om die erbij te hebben. In de eerste plaats is het

makkelijker om een bepaalde rij aan te duiden. Er zouden bijvoorbeeld twee jongens in een klas kunnen zitten die allebei Jan de Vries heten. Als je één van de twee wilt aanduiden, is het niet voldoende om alleen zijn voornaam, het tussenvoegsel en zijn achternaam op te geven. Je moet dan ook zijn straat en huisnummer opgeven. Door een leerlingnummer in de tabel op te nemen, kun je met alleen dat nummer volstaan. Je moet er dan wel voor zorgen dat al die nummers verschillen. Dat is gelukkig niet zo moeilijk want een relationeel systeem is daar goed voor uitgerust. Elke keer dat er iemand in het systeem wordt opgenomen, geeft de administrateur een nieuw leerlingnummer.
Een goed programma doet dat automatisch en zorgt er meteen voor dat er alleen maar unieke nummers worden gebruikt. Een tweede reden wordt verderop in dit hoofdstuk pas echt duidelijk. Bij wat complexere vragen staan de gegevens in verschillende tabellen. Je wilt bijvoorbeeld weten wie welke boeken hebben geleend. De gegevens over de leerlingen staan in de tabel Leerlingen, de gegevens over de boeken staan in de tabel Boeken en de gegevens over uitleningen staan in de tabel Uitleningen. We zullen dan uit drie tabellen de informatie moeten halen. Om dat wat doelmatiger te laten gebeuren, zorgen we ervoor dat elke rij in die tabellen eenvoudig is te identificeren. Dat kan natuurlijk met voornaam, tussenvoegsel, achternaam, straat en huisnummer. Maar dat zijn vijf kolommen! Als je iedere leerling een uniek nummer geeft, heb je alleen maar dat ene leerlingnummer nodig. Niet alleen scheelt dat heel wat typwerk, en dus heel wat typfouten, maar het komt natuurlijk de snelheid van verwerken ook ten goede.

3 EEN EERSTE SQL-QUERY

Een SQL-query kent een vaste structuur. Je moet aangeven over welke tabel het gaat en welke kolommen en rijen je te zien wilt krijgen. In SQL kun je die vaste structuur herkennen aan de sleutelwoorden. Die sleutelwoorden kun je het best steeds vooraan op een nieuwe regel zetten. Het is niet strikt noodzakelijk, maar het wordt er wat overzichtelijker door. De basisstructuur van de meest eenvoudige query is:

```
SELECT    <een of meer kolommen>
FROM      <een of andere tabel>
```

Het is voor de computer volstrekt onbelangrijk of wij elk sleutelwoord op een nieuwe regel beginnen. Voor de computer ziet de voorgaande query er zo uit:

```
select * from leerlingen where achternaam = 'bakker' and tussenvoegsel is null and voornaam ='jantine'
```

Het is een lange worst van tekens. Je kunt je voorstellen dat het wel erg onoverzichtelijk wordt als de query nog langer is. Daarom noteren we de query zo dat hij overzichtelijk en leesbaar is. Elk sleutelwoord op een nieuwe regel en om het nog overzichtelijker te maken, schrijven we de sleutelwoorden vaak in hoofdletters. Dat is overigens niet nodig: SQL is **niet** hoofdlettergevoelig.

We zouden om te beginnen eens kunnen kijken welke leerlingen in deze database zitten en wat er van hen bekend is. De SQL-query ziet er dan zo uit:

```
SELECT    llnr, voornaam, tussenvoegsel, achternaam, straat, huisnummer,
          postcode, plaats, telefoon, geslacht, geb_datum, klas
FROM      leerlingen
```

Achter SELECT geef je de kolommen op die je in de uitvoer wilt zien, gescheiden door komma's. Het is onhandig als je al die kolomnamen moet invoeren. Bovendien

luistert het ook nog eens erg nauw. Als je een typfout maakt, krijg je een foutmelding. Er is een manier om deze SQL-query wat eenvoudiger te maken. Als je alle kolommen wilt hebben, dan is het voldoende als je een sterretje invult. Achter FROM geef je de naam van de tabel (of tabellen) waaruit deze kolommen komen. Dezelfde SQL-query ziet er dan zo uit:

```
SELECT    *
FROM      leerlingen
```

Door het sterretje krijg je alle kolommen van de tabel op het scherm. In een standaard SQL-omgeving heb je query's zoals deze nodig om de inhoud van tabellen te zien te krijgen. In Access kun je ook onder de tab Tabellen in het databasevenster een tabel openen. Hieronder een tabel van Access:

LEERLINGEN : Tabel

LLN	VOORNAA	TUSSEN	ACHTERNAAM	STRAAT	HUISNU	POSTCO	PLAATS	TELEFOON	GES	GEB_DATUM	KLAS
51	Janco		Dijke van	Houtenseweg	8	3984LJ	Odijk	030-6542278	m	16-sep-1989	5a
52	Rene		Bokker	Graaf Boudewijnlaan	35	3464DV	Nieuwegein	030-6014019	m	20-feb-1991	5b
53	Marco	van	Bemmel	Wielingenstraat	6	3522PG	Utrecht	030-2112237	m	6-okt-1990	4a
54	Petra		Rohstein	Vlierweg	40	3991BE	Houten		v	29-jan-1991	4b
55	Nourdine		Abloukie	Potgieterstraat	25 bis	3532VP	Utrecht	030-2864548	v	23-sep-1989	5a
56	Nathalie	van	Horp	Auriollaan	71	3527ES	Utrecht	030-2231231	v	13-mrt-1988	6b
57	Debby	van der	Boogaard	Ferdinand Bolstraat	70	3586AS	Utrecht	030-2456132	v	19-apr-1990	5b
58	Monique		Aalbers	Lange Uitweg	18	3998WE	Schalkwijk	030-6035676	v	20-jun-1989	6b
59	Yvanka		Uittenbogaard	Huize de Geerlaan	121	3523JN	Utrecht		v	21-jan-1989	6a
60			Bosveld	Boslaan	16	3981XD	Bunnik	030-6554654	v	2-aug-1988	6b

Bedenk wel dat een normale gebruiker van een database nooit zomaar in de tabellen kan kijken of werken; dat is vrijwel altijd afgeschermd.

Kolommen kiezen
Je krijgt erg veel informatie als je met het sterretje alle kolommen opvraagt. Vaak wil je alleen maar namen en bijvoorbeeld de klas waar iedereen in zit. Je moet dan gewoon opgeven welke kolommen je wilt hebben. De volgorde waarin je de kolommen vermeldt, is van belang; de kolommen komen in die volgorde op het scherm

```
SELECT    voornaam, tussenvoegsel, achternaam, klas
FROM      leerlingen
```

Sorteren
Om er wat ordening in te brengen kunnen we de resultaattabel laten sorteren. Dat zouden we kunnen doen op achternaam. We moeten de SQL-query dan uitbreiden met een regel ORDER BY:

```
SELECT    voornaam, tussenvoegsel, achternaam, klas
FROM      leerlingen
ORDER BY  achternaam
```

Bij het sorteren wordt uitgegaan van de ASCII-representatie van de tekens die in de kolom staan. Dus eerst de cijfers 0 t/m 9, dan de hoofdletters A t/m Z en als laatste de kleine letters a t/m z. Je kunt ook op twee kolommen laten sorteren, bijvoorbeeld eerst op Klas en daarna op Achternaam. De uitvoertabel is dan gesorteerd op klas en binnen elke klas wordt er gesorteerd op achternaam. Sorteren op twee kolommen doe je door achter ORDER BY de twee kolommen op te geven, gescheiden door een komma. De volgorde waarop je de kolommen noemt, maakt verschil!
Je kunt de kolommen waarop je sorteert ook anders aanduiden. In plaats van de naam van deze kolom noem je dan het volgnummer van de kolom. Dat vind je door

naar de volgorde te kijken waarin ze achter SELECT staan vermeld.

```
SELECT    voornaam, tussenvoegsel, achternaam, klas
FROM      leerlingen
ORDER BY  4,3
```

Dat is met name van belang als het gaat om functies in een query (komt later aan de orde). Ten slotte kun je op twee manieren sorteren: oplopend en aflopend. In het Engels is dat ascending en descending, afgekort tot ASC en DESC. Als je niets vermeldt, wordt automatisch oplopend gesorteerd. Je laat dus aflopend sorteren door dit achter de kolomaanduiding na ORDER BY te vermelden.

```
SELECT    voornaam, tussenvoegsel, achternaam, klas
FROM      leerlingen
ORDER BY  3 DESC
```

Alleen verschillende
Als we een overzicht willen hebben van de woonplaatsen van de leerlingen in het systeem, maken we de volgende query:

```
SELECT    plaats
FROM      leerlingen
```

Het resultaat is een lange waslijst van plaatsnamen die allemaal een aantal keren voorkomen. Om precies te zijn, we selecteren uit de leerlingentabel en daardoor zal een plaats waar tien leerlingen wonen ook tien keer in de tabel staan, voor elk van de leerlingen één keer. Om het een beetje overzichtelijk te houden willen we elke plaats maar één keer in het scherm hebben. Het programma moet de dubbele rijen verwijderen. We passen daarvoor de vraag aan met het sleutelwoord DISTINCT.

```
SELECT    DISTINCT plaats
FROM      leerlingen
```

Nu krijgen we een keurig ingekorte lijst van woonplaatsen. We zouden deze eventueel nog kunnen laten sorteren.

➔ **Opdrachten**

1. Maak een lijst val alle leerlingen, waarbij de achternaam vooraan staat. Sorteer de lijst oplopend op achternaam.
2. Maak een lijst van alle leerlingen in het systeem gesorteerd op klas en binnen de klas op achternaam
3. Maak een lijst van alle auteurs gesorteerd op geboortejaar en achternaam
4. Maak een lijst van alle boeken gesorteerd op rubriek en titel
5. Maak een lijst van alle boeknummers die ooit zijn uitgeleend. Sorteer in oplopende volgorde.
6. Maak een lijst van alle klassen die in de database voorkomen.

4 VOORWAARDEN IN EEN SQL-QUERY

We hebben gezien dat we de kolommen uit een tabel kunnen selecteren door deze achter de instructie SELECT te vermelden. Zo is het mogelijk om niet alle rijen uit een tabel op te roepen, maar om daar ook een selectie te maken. Dat doen we door voorwaarden te stellen aan de rijen, in een extra regel van de query. De basisstructuur van een query wordt nu:

```
SELECT    <een of meer kolommen>
FROM      <een of andere tabel>
WHERE     <een of meer voorwaarden die moeten gelden>
```

We willen bijvoorbeeld een lijst van alle jongens hebben.

```
SELECT    voornaam, tussenvoegsel, achternaam
FROM      leerlingen
WHERE     geslacht = 'm'
```

Voorwaarden geef je op achter het sleutelwoord WHERE. De gestelde voorwaarde is dat er in de kolom Geslacht een letter 'm' staat. Je selecteert uit de tabel Leerling
! dus de kolommen die achter SELECT staan en van de rijen in die kolommen alleen die rijen die een 'm' hebben in de kolom Geslacht.
SQL maakt onderscheid tussen verschillende datatypen. Meestal gaat het om gewoon tekst of woorden die alleen bedoeld zijn om te lezen. In die gevallen moeten er altijd aanhalingstekens worden gebruikt. Daarom staat er Geslacht = 'm'. Wees er zorgvuldig mee, want SQL is onbarmhartig waar het om typfouten gaat! Er wordt letterlijk gekeken of elk teken dat je invoert exact overeenkomt met wat er in de tabel staat.
Als er in een kolom cijfers staan, zijn er twee mogelijkheden. Telefoonnummers en
! huisnummers (bijvoorbeeld '12bis') zijn geen getallen waarmee je kunt rekenen. Daarom worden die behandeld als tekst, zoals Achternaam. Je moet dan aanhalingstekens gebruiken. Als je een kolom hebt met getallen waarmee je kan rekenen, dan mag je juist geen aanhalingstekens gebruiken. In de bibliotheekdatabase bevat de kolom Boete in Uitleningen getallen waarmee je kunt rekenen, je kunt bijvoorbeeld het totaal van de boetes uitrekenen.

Maskers
De exactheid bij het vergelijken is soms lastig, bijvoorbeeld omdat je de spelling van een naam niet precies weet. Stel, je hebt van een boek een uittreksel nodig en je weet dat er iemand is die er een heeft. Je zit er dringend om verlegen en wilt vragen of je het mag kopiëren. Maar je weet niet precies wie het heeft. Je weet dat hij of zij Frederiks heet, maar het zou ook Frederics of zelfs Frederikse kunnen zijn. Het enige dat je weet is dat de naam klinkt als Frederiks. Het is dan twijfelachtig of de volgende query iets oplevert:

```
SELECT    voornaam, tussenvoegsel, achternaam, telefoon
FROM      leerlingen
WHERE     achternaam = 'frederiks'
```

Het probleem zit in de =. Er wordt teken voor teken gekeken of de achternamen in de tabel gelijk zijn aan de opgegeven naam. We zouden liever zien dat alles geselecteerd wordt wat ongeveer klopt, en dan zien we zelf eventueel wel of we de vraag nog moeten verfijnen.
Hiervoor gebruiken we dan ook niet de = maar het sleutelwoord LIKE. Daardoor kunnen we aangeven welke letters precies moeten kloppen en wat ongeveer moet kloppen. Het procentteken % gebruiken we als masker als er van alles mag staan. Het mag een enkel teken zijn, maar vijftien tekens mag ook. Als je precies weet hoeveel tekens er moeten staan, maar je weet niet welke tekens, dan kun je de underscore_ als masker gebruiken voor elk van die tekens. Microsoft Access wijkt hier af van de standaard: het gebruikt * in plaats van % en ? in plaats van _.

```
SELECT    voornaam, tussenvoegsel, achternaam, telefoon
FROM      leerlingen
WHERE     achternaam LIKE 'frederi%'
```

Deze query selecteert alle achternamen waarvan de eerste zeven tekens 'Frederi' zijn. De tekens die daarna komen worden niet vergeleken.

```
SELECT    voornaam, tussenvoegsel, achternaam, telefoon
FROM      leerlingen
WHERE     achternaam LIKE 'freder_s'
```

Met deze query worden wel Frederiks en Frederics geselecteerd, maar Fredericks en Frederikse en Frederiksen niet. Alléén het teken waar de underscore staat wordt niet vergeleken, de rest moet exact overeenkomen.
We noemen = en LIKE operatoren. Er zijn nog meer operatoren, ze zijn bekend uit de wiskunde en werken vrijwel hetzelfde. Ze werken op de een of andere manier allemaal op basis van een getalswaarde. Bij woorden is dat de ASCII-representatie en bij getallen is dat de getalswaarde. De complete lijst:

=	Is precies gelijk aan.
LIKE	Is ongeveer gelijk aan. Zet in de tekst een % (Access: *) voor een willekeurige reeks onbekende tekens, _ (Access: ?) voor één onbekend teken.
< >	Kleiner dan en groter dan. Bij teksten is dat ervoor of erna in alfabetische volgorde.
<= >=	Kleiner dan of gelijk aan' en 'groter dan of gelijk aan'. Let op de volgorde: de = moet achteraan staan.
< >	Is niet gelijk aan'. De schrijfwijze verschilt per systeem. Soms is het Λ=, != 0 of ≠.

➔ Opdrachten

7. Welke leerlingen hebben de achternaam Dijkstra?
8. Welke schrijvers zijn na 1900 geboren?
9. Welke boeken vallen niet in de rubriek Nederlands?
10. Gerard Reve is een bekende auteur. Een aantal jaren geleden heeft hij zijn naam veranderd, vroeger heette hij Gerard van het Reve. Welke boeken zijn van hem in de bibliotheek?
11. Zijn er boeken in de bibliotheek van William Somerset Maugham?
12. Stel de query op waarmee je kunt zoeken naar leerlingen met een naam die klinkt als Frederiks. Welke namen levert dit op?
13. Hoeveel leerlingen hebben de lettercombinatie 'an' in hun voornaam?
14. Welke boeken in de bibliotheek gaan over Reinaart de Vos? Denk eraan, deze spelling is modern en het boek dateert uit de middeleeuwen.

5 SAMENGESTELDE VOORWAARDEN IN EEN SQL-QUERY

Sommige vragen zijn niet zo eenvoudig te beantwoorden. Bijvoorbeeld, welke klasgenoten van Jantine Bakker wonen bij haar in de buurt? Voordat je de query kunt opstellen, moet je eerst uitzoeken waar Jantine woont en in welke klas ze zit. (Later bekijken we query's waarmee je in één keer leerlingen die bij haar in de buurt wonen vindt.)

```
SELECT    *
FROM      leerlingen
WHERE     achternaam = 'bakker'
```

Uit de lijst van de familie Bakker zoeken we nu uit waar Jantine precies woont. Maar

dat zou lastig kunnen zijn als er een Jan Bakker en een Josien de Bakker en een Joris de Bakker en een Petra Bakker in het systeem zitten! We moeten de vraag dus wat aanscherpen. De achternaam moet 'Bakker' zijn en de voornaam Jantine. Hiervoor gebruik je het sleutelwoord AND:

```
SELECT    *
FROM      leerlingen
WHERE     achternaam = bakker'
AND       voornaam = 'jantine'
```

Maar we zouden toch nog een probleem kunnen krijgen als we Jantine Bakker willen hebben en er is ook nog een Jantine de Bakker. Je krijgt namelijk beide als je de vraag zo hebt geformuleerd.
Het verschil zit in het tussenvoegsel: de een heeft het wel en de ander niet. We willen Jantine Bakker hebben zonder 'de'. Er moet dus in de kolom Tussenvoegsel niets ingevuld staan. Dit is iets speciaals. Voor het programma bestaat er een groot verschil tussen 'niets ingevuld' en een spatie, die je op het scherm niet ziet.
! In SQL betekent NULL 'niet ingevuld'. Je zoekt naar niet-ingevulde waarden met de voorwaarde '<kolomnaam> IS NULL'. Let op de operator IS. Je mag niet de operator = gebruiken, dat is in SQL gereserveerd voor de letterlijke vergelijking (iemand die NULL heet!). De query is dus:

```
SELECT    *
FROM      leerlingen
WHERE     achternaam = 'bakker'
AND       tussenvoegsel IS NULL
AND       voornaam = 'jantine'
```

Naast AND is er ook het sleutelwoord OR, of. Hiermee kun je bijvoorbeeld de leerlingen zoeken die in Bunnik of Houten wonen.

```
SELECT    *
FROM      leerlingen
WHERE     plaats = 'bunnik'
OR        plaats = 'houten'
```

! De OR is niet exclusief. Dat wil zeggen dat de rijen die aan beide voorwaarden voldoen ook altijd worden geselecteerd. De volgende query geeft de gegevens van de leerlingen die in klas 4b zitten en daar bovenop de gegevens van de leerlingen die in Utrecht wonen. Daarbij zitten 4b'ers uit Houten, Utrechters uit klas 5v, maar ook de Utrechters uit klas 4b.

```
SELECT    *
FROM      leerlingen
WHERE     plaats = 'utrecht'
OR        klas = '4b'
```

Er wordt een bovenbouwfeest gegeven. Daar mogen uiteraard alleen leerlingen van klas 4 en hoger komen. Jantine Bakker is nieuw in klas 4b en ze wil er graag heen. Thuisgebracht worden zal geen probleem zijn, maar ze mag er van haar ouders niet alleen naartoe fietsen. Ze moet een paar meisjes vinden die met haar mee fietsen. Liefst meisjes uit haar klas, maar als dat niet lukt, mogen ze ook best in een andere klas zitten, als ze maar dicht in de buurt wonen.

Dat is een complexe voorwaarde: het gaat om meisjes en ze moeten bij haar in de

klas zitten óf ze moeten in de buurt wonen. Het lijkt eenvoudig, want ‘en of’ vertaal je gewoon met AND en OR. Maar er zitten enkele stevige adders onder het gras. Dat komt omdat je AND en OR samen in één voorwaarde gebruikt.
Als je een wat oudere of heel simpele calculator gebruikt, krijg je soms een soortgelijk probleem. Probeer maar eens uit te rekenen: 1 + 2 x 3 of 2 x 3 + 4 x 5. Een goede calculator geeft de antwoorden 7 en 26, maar een erg eenvoudig gevalletje geeft 9 en 50. De oorzaak zit in een rekenregel: vermenigvuldigen gaat vóór optellen. Eenvoudige calculators kennen die regel niet en dat geeft onjuiste antwoorden. Wil je dan toch het goede antwoord hebben, dan kun je bijvoorbeeld haakjes gebruiken: 1 + (2 * 3) en (2 * 3) + (4 * 5). Een soortgelijk probleem doet zich voor bij voorwaarden waarin zowel AND als OR voorkomen. Een voorwaarde die er uitziet als '... AND... OR...' zou op twee manieren gelezen kunnen worden: als '(... AND...) OR...' en als '... AND (... OR...)'. SQL doet het eerste. In een voorwaarde met AND en OR gaat de AND dus voor. Vaak is zo'n voorwaarde nogal verwarrend omdat deze groot is. Daarom is het verstandig toch haakjes te zetten. Bovendien, een complexe voorwaarde wordt overzichtelijker als je hem zorgvuldig noteert en haakjes gebruikt.

!

Jantine moet een paar meisjes vinden die met haar mee fietsen. Liefst uit haar klas, maar als dat niet lukt, mogen ze ook best in een andere vierde klas zitten, als ze dan maar dicht in de buurt wonen.

In SQL wordt dit:

```
SELECT      voornaam, tussenvoegsel, achternaam, straat, huisnummer,
            postcode, plaats, telefoon
FROM        leerlingen
WHERE       geslacht = ‘v’
    AND     ( klas = '4b'
            OR  ( klas LIKE ‘4_’
            AND postcode LIKE '3523%’)
            )
```

Je ziet dat in een samengestelde voorwaarde best verschillende niveaus van haakjes kunnen voorkomen. Het resultaat van de query wordt anders als je de haakjes weglaat.

➔ Opdrachten

15. Welke leerlingen wonen in Utrecht aan de Julianaweg?
16. Welke leerlingen uit de zesde klas zijn na 1 januari 1980 geboren?
17. Welke leerlingen wonen in Bunnik of Schalkwijk?
18. Voor deze query uit:

```
select      *
from        leerlingen
where       klas = ‘5b’
      and   plaats = ‘Utrecht’
       or   plaats = ‘Bunnik’
```

a In het resultaat zullen ook leerlingen voorkomen die niet in klas 5b zitten. Hoe komt dat? Sorteer op klas
b Geef aan hoe SQL de haakjes zet in deze query.
c Welke query moet je uitvoeren om de leerlingen uit klas 5b te vinden die in Utrecht of Bunnik wonen?

19. Op het schoolterrein is een fiets gevonden met een ingegraveerde postcode die maar gedeeltelijk leesbaar is. De postcode begint met de cijfers 352. De rest is niet te lezen, behalve de laatste letter, dat is een B, R of P. De fiets is vermoedelijk van iemand van school. Van wie zou die fiets kunnen zijn?
20. Deze opgave borduurt door op het probleem van Jantine. Jantine wil bij het bovenbouwfeest op safe spelen en alvast uitzoeken welke jongens haar wellicht naar huis zouden mogen brengen. Daar gelden uiteraard strenge criteria voor, het mag thuis geen problemen oproepen.
    De kandidaten moeten natuurlijk aardig en cool zijn, maar ook wel ouder dan Jantine. Ook weer niet te oud, een jaar ouder is in orde. Ze mogen eventueel ook nog wel twee jaar ouder zijn, maar dan moeten ze dicht in de buurt wonen. Wie staan er op de short list?

Data hebben in Access een bijzondere notatie: **#1/1/1980#**

6 FUNCTIES IN EEN SQL-QUERY

Aan het eind van het schooljaar wil de directie weten hoeveel boeken er in de bibliotheek staan. Dat lijkt een eenvoudige vraag, maar wat bedoelen ze eigenlijk? Boeken of exemplaren? Of allebei, per boek het aantal exemplaren? Gewoon stuk voor stuk tellen is niet handig en bovendien zijn er natuurlijk boeken uitgeleend, in reparatie of gewoon kwijt. We laten daarom de computer tellen hoeveel boeken er zouden moeten zijn.
Voor het tellen heeft SQL een oplossing. Bijvoorbeeld, als je gewoon wilt weten hoeveel boeken er in de bibliotheek zouden moeten zijn:

```
SELECT    count (*)
FROM      exemplaren
```

Het resultaat is een enkel getal dat aangeeft hoeveel rijen er in de tabel Exemplaren zijn. COUNT is een functie die dit voor je uitvoert. Mogelijke functies zijn:

COUNT(*)	telt het aantal rijen in een tabel.
SUM(kolomnaam)	hiermee bereken je de som van een aantal getallen in een kolom. Let op, dit is iets anders dan het aantal rijen in een kolom, hierbij wordt er opgeteld en niet alleen geteld!
MAX(kolomnaam)	levert de grootste waarde in een kolom. Het maakt niet uit of het een kolom getallen of een kolom woorden is.
MIN(kolomnaam)	levert de kleinste waarde in een kolom. Het maakt niet uit of het een kolom getallen of een kolom woorden is.
AVG(kolomnaam)	berekent het gemiddelde van de getallen in een kolom. Natuurlijk kan dat alleen als er getallen in die kolom staan.

Het totaalbedrag van de boetes op uitleningen kun je berekenen met:

```
SELECT    SUM(boete)
FROM      uitleningen
```

Groeperen

Met de functie COUNT is het eenvoudig om van elke klas te weten te komen hoeveel leerlingen erin zitten.

```
SELECT    COUNT(*)
FROM      leerlingen
WHERE     klas = '4a'
```

```
SELECT    COUNT(*)
FROM      leerlingen
WHERE     klas ='4b'
```

```
SELECT    COUNT(*)
FROM      leerlingen
WHERE     klas = '5a'
```

Maar het is natuurlijk onhandig om dat voor elke klas te doen. Het ligt voor de hand om dat in een enkele query te doen:

```
SELECT    klas, COUNT(*)
FROM      leerlingen
```

Toch krijg je een foutmelding als je het zo probeert. Dat komt omdat wij zelf wel begrijpen wat we bedoelen, maar voor het programma is het niet duidelijk wat er nu geteld moet worden. Alle leerlingen? Alle klassen? Of alleen de leerlingen in een klas? Eigenlijk geeft de vraag ook niet goed weer wat we willen. We willen immers dat er per klas geteld wordt en dat staat niet in de query. We zouden willen dat er eerst groepjes gemaakt worden van de verschillende klassen en dat vervolgens per groep geteld wordt. Het resultaat van de query zou dan een lijstje moeten zijn van de verschillende klassen met achter elke klas het aantal leerlingen.
Om te groeperen heeft SQL de opdracht GROUP BY. De query wordt dan:

```
SELECT    klas, COUNT(*)
FROM      leerlingen
GROUP BY  klas
```

Let op: achter GROUP BY moeten alle kolommen genoemd worden die ook achter SELECT staan!

! Dit is ook de enige manier om een kolom en een functie te combineren in de SELECT-regel: met een extra regel met GROUP BY <KOLOMNAAM>.

➔ Opdrachten

21. Wat is de grootste boete die iemand ooit heeft moeten betalen?
22. Wat is de gemiddelde boete die werd opgelegd, gerekend over alle uitleningen?
23. Wat is de geboortedatum van de oudste leerling van de school? En van de jongste?
24. Hoeveel boeken zijn er in reparatie?
25. Maak een lijstje van aantallen leerlingen per woonplaats
26. Hoeveel jongens en hoeveel meisjes zitten er op school?
27. Maak een overzicht van de aantallen jongens en meisjes per klas. Zijn er klassen waar de verhouding erg scheef is?
28. Hoeveel boeken zijn er van elke rubriek aanwezig?

7 DE SUBQUERY

De directie wil inzicht in de boetes die de leerlingen moeten betalen als ze een boek te laat inleveren. Als die te laag zijn, trekken jongelui zich niet zo veel aan van de uitleentermijn, maar als ze te hoog zijn, kan dat een drempel opwerpen om van de bibliotheek gebruik te maken. Daarom moet eerst eens worden uitgezocht welke leerlingen de meeste boete hebben betaald toen ze een boek terugbrachten.
Dat zijn eigenlijk twee vragen: hoe hoog was de maximale boete en wie moest die betalen. De eerste vraag is het eenvoudigst:

```
SELECT    MAX(boete)
FROM      uitlening
```

We kunnen nu het bedrag van de boete (dat was € 4,00) noteren en vervolgens uitzoeken wie de boosdoener was. Maar hier stuiten we op een probleem. We hebben nu wel de grootste boete, maar de tabel UITLENING kent alleen leerlingnummers en geen namen. We moeten dus eerst uitzoeken welk leerlingnummer bij een boete van € 4,00 hoort en dan van wie dat nummer is.

```
SELECT    llnr
FROM      uitleningen
WHERE     boete = 4.00
```

Deze twee query's kunnen we combineren tot een enkele query.
Daarvoor vervangen we de uitkomst 4,00 door de query die dit resultaat opleverde:

```
SELECT    llnr
FROM      uitleningen
WHERE     boete =
          (  SELECT    MAX(boete)
             FROM      uitleningen
          )
```

In plaats van het bedrag € 4,00 is de hele query ingevuld die dat bedrag heeft opgeleverd. We spreken hiervan een **subquery**. We hebben nu het leerlingnummer, maar nog geen naam en klas. We breiden de vraag dus weer verder uit door de vorige query de subquery te maken in een nieuwe query:

```
SELECT    voornaam. achternaam. klas
FROM      leerlingen
WHERE     llnr =
          ( SELECT    llnr
            FROM      uitleningen
            WHERE     boete =
                      (SELECT     MAX(boete)
                      FROM        uitleningen
                      )
          )
```

Toch kan hier nog een en ander fout gaan. Het zou best kunnen dat er drie leerlingen zijn die een boete van € 4,00 hebben moeten betalen. Als zich dat voordoet, krijgen we een foutmelding. De voorwaarde die we stellen luidt namelijk dat het leerlingnummer gelijk moet zijn aan een bepaald nummer. Maar de subquery levert niet een enkel nummer op, maar drie verschillende nummers. Als we dat willen toelaten, moeten we niet de = gebruiken. Die eist immers dat wat er links en rechts van staat precies gelijk is. En één leerlingnummer links is natuurlijk niet hetzelfde als drie verschillende nummers rechts.
We lossen dit op door de = te vervangen door IN. Nu kan het programma het wel aan, want nu wordt van elk LLNR gecontroleerd of het wellicht in het rijtje van drie nummers voorkomt die de subquery heeft opgeleverd.

```
SELECT      voornaam, achternaam. klas
FROM        leerlingen
WHERE       llnr IN
            (SELECT     llnr
            FROM        uitleningen
            WHERE       boete =
                        (SELECT     MAX(boete)
                        FROM        uitleningen
                        )
            )
```

Bij dit soort problemen moet je gebruikmaken van verschillende tabellen tegelijk, omdat de gegevens zo zijn geordend. Het is verstandig om dan de tabellenstructuur van de database bij de hand te houden. Ga eerst voor jezelf na wat je eigenlijk wilt weten. Dat moet in de SELECT-regel van de hoofdquery. Vervolgens ga je onderzoeken wat je nodig hebt om aan de voorwaarden die achter WHERE komen te voldoen. Probeer niet meteen de hele query in één keer op te zetten, experimenteer gerust met stukken ervan zoals in het voorbeeld hiervoor. Als het complex, is kun je het best eenvoudig beginnen en de vraag steeds verder uitbreiden.

Hierover is een heel instructieve powerpoint-presentatie beschikbaar.

➔ Opdrachten

29. Wie is de oudste leerling van de school? En wie de jongste?
30. Welke boeken in de rubriek Nederlands zijn in reparatie?
31. Van welke auteurs werden er boeken gereserveerd?
32. Een boek werd beschadigd in de bibliotheek teruggevonden. Het exemplaarnummer is zelfs niet meer te lezen. Het gaat om een exemplaar van '*Au pair*' van W.F. Hermans. Wie zou dat op zijn/haar geweten kunnen hebben?

8 DE JOIN

De gegevens in een relationeel informatiesysteem zijn over verschillende tabellen verdeeld. Daardoor komt het voor dat je gegevens uit verschillende tabellen tegelijk wilt hebben. SQL voorziet in een mogelijkheid om tabellen willekeurig te combineren. Er zit een keerzijde aan, want die combinatietabel kan verschrikkelijk groot worden. Dat komt de verwerkingssnelheid natuurlijk niet ten goede.
Als voorbeeld een kleine database van twee tabellen met elk drie rijen:

TABEL JONGENS			TABEL MEISJES	
NAAM	WOONPLAATS		NAAM	WOONPLAATS
Jan	Leiden		Inge	Leiden
Piet	Assen		Marleen	Assen
Kees	Arnhem		Joyce	Arnhem

Als we nu een combinatie van deze twee tabellen maken, gaat dat heel eenvoudig.

```
SELECT      *
FROM        jongens, meisjes
```

Door de twee tabellen gescheiden door een komma achter FROM te zetten krijgen we de combinatie al. We spreken hier van een koppeling van tabellen, in het Engels **join** (to join = samenvoegen). Maar wat zit er in die combinatie?

NAAM	WOONPLAATS	NAAM	WOONPLAATS
Jan	Leiden	Inge	Leiden
Jan	Leiden	Marleen	Assen
Jan	Leiden	Joyce	Arnhem
Piet	Assen	Inge	Leiden
Piet	Assen	Marleen	Assen
Piet	Assen	Joyce	Arnhem
Kees	Arnhem	Inge	Leiden
Kees	Arnhem	Marleen	Assen
Kees	Arnhem	Joyce	Arnhem

! Je ziet dat elke rij uit de tabel Jongens gecombineerd wordt met elke rij uit de tabel Meisjes. De combinatie van twee tabellen van elk drie rijen geeft dus een combinatietabel van negen rijen! Als er in de ene tabel 100 rijen zitten en in de andere 200, dan heeft de combinatie 100 x 200 = 20.000 rijen. Bedenk eens hoeveel leden de ANWB heeft, of hoeveel cliënten een verzekeringsmaatschappij. Als daar combinaties gemaakt worden, moet er heel voorzichtig mee worden omgegaan alleen al vanwege de enorme combinatietabellen die zouden kunnen ontstaan.
Je heb niets aan die grote combinatietabel, want je moet natuurlijk wel de juiste rijen aan elkaar koppelen. Niet zomaar alles aan alles, maar goed kijken wat je wilt hebben. Voor ons is het belangrijk eens goed te kijken naar de meest doelmatige combinatie. Dus niet zomaar alles met alles. Stel dat je een lijst wilt van jongens en meisjes die in dezelfde stad wonen. Dan koppel je juist die rijen aan elkaar waar de woonplaats hetzelfde is. Je voegt er dus gewoon een voorwaarde aan toe die dat voor je regelt.

```
SELECT    *
FROM      jongens, meisjes
WHERE     woonplaats = woonplaats
```

Helaas, dat levert een foutmelding op. Eigenlijk ligt dat ook wel een beetje voor de hand. Er wordt twee keer een tabel WOONPLAATS genoemd in de query. Hoe zou dat programma nou moeten weten welke er bedoeld wordt? We passen de query dus maar aan:

```
SELECT    *
FROM      jongens, meisjes
WHERE     jongens.woonplaats = meisjes.woonplaats
```

Nu gaat het wel goed en we krijgen een keurige lijst zoals we die willen hebben.

NAAM	WOONPLAATS	NAAM	WOONPLAATS
Jan	Leiden	Inge	Leiden
Piet	Assen	Marleen	Assen
Kees	Arnhem	Joyce	Arnhem

Over de join is een instructieve powerpoint-presentatie beschikbaar.

Die woonplaats hoeft niet twee keer vermeld te worden. Die kolomnamen zijn soms wat aan de lange kant. Je kunt in de query een afkorting voor de tabelnamen opnemen. Heel simpel, je zet er gewoon een letter achter. Dit heet een **pseudoniem**.

```
SELECT    j.naam, m.naam, j.woonplaats
FROM      jongens j, meisjes m
WHERE     j.woonplaats = m.woonplaats
```

In beide tabellen staat een kolom Naam. Als je ze allebei wilt oproepen, moet je dus ook weer de tabelnaam erbij vermelden. Gelukkig kan dat ook met de letters J en M. Merkwaardig is dat in de regel achter SELECT al gebruik wordt gemaakt van die afkortingen terwijl ze pas in de regel achter FROM gedeclareerd worden. Kennelijk begint het programma de regel met FROM als eerste te lezen.

Nu een voorbeeld uit de bibliotheek. Elke week wil de bibliothecaris een overzicht van alle boetes. Een lijst met voornaam, achternaam, klas en datum en bedrag van de boete.
Daarvoor moeten de tabellen Leerlingen en Uitleningen worden gecombineerd. Het verband tussen de twee tabellen zit in de kolommen Llnr. In de tabel Uitleningen zit ook een kolom Llnr. Die geeft aan welke leerling het boek geleend heeft, het is een verwijzing naar de leerlingentabel. Er komt een voorwaarde bij, tegelijk passen we de pseudoniemdeclaratie toe:

```
SELECT   voornaam, tussenvoegsel, achternaam, datum_uit, datum_terug, boete
FROM     leerlingen l, uitleningen u
WHERE    l.llnr = u.llnr
```

Meestal gaat het om twee kolommen die dezelfde of een soortgelijke inhoud hebben. Het is niet noodzakelijk dat ze ook dezelfde naam hebben. Woonplaats, Gemeente en Plaats zouden kolomnamen kunnen zijn die precies dezelfde inhoud hebben. Soms staat de informatie die je nodig hebt over drie of zelfs meer tabellen verspreid. In dat geval moet je een join maken van meer dan twee tabellen en bij de voorwaarde de tabellen twee aan twee koppelen.

➔ Opdrachten

33. Maak een lijst van auteurs en de titels van hun boeken die in het bibliotheekbestand zitten
34. Maak een lijst van leerlingen en de boetes die ze hebben betaald. Zorg dat de uitleningen zonder boete (in de tabel staat dan 0,00) niet in de lijst voorkomen
35. Welke boeken zijn er in de bibliotheek van schrijvers uit de negentiende eeuw?
36. Van welke auteurs zijn er boeken in reparatie?
37. Maak een overzicht van alle boeken uit de rubriek Nederlands die uitgeleend zijn aan leerlingen van de zesde klas. Zorg ervoor dat de naam van de leerling en de titel en de auteur in het overzicht staan.

9 GROUP BY

Van sommige boeken zijn er verschillende exemplaren en van andere maar een enkel exemplaar. We willen een lijst maken van alle boeken, compleet met auteursnaam, waarvan er twee of meer exemplaren in de bibliotheek aanwezig zijn. Daarvoor moeten we eerst een lijst maken van alle boeken en het aantal exemplaren dat er is.

```
SELECT      a.voornaam, a.achternaam, b.titel, count(exnr)
FROM        auteurs a, boeken b, exemplaren e
WHERE       a.auteurnr = b.auteurnr
      AND   e.boeknr = b.boeknr
GROUP BY    a.voornaam, a.achternaam, b.titel
```

Omdat deze lijst erg lang is, stellen we de extra voorwaarde dat we alleen die boeken willen hebben waarvan er minstens twee exemplaren zijn. Die voorwaarde wordt niet aan rijen opgelegd die achter SELECT staan, maar aan de groepjes die deze query oplevert. Daarvoor moeten we de query uitbreiden met een extra regel waarin dit wordt geregeld. Hiervoor is de HAVING-regel die achteraan wordt toegevoegd. Met HAVING kun je voorwaarden stellen aan de groepjes die je met GROUP BY hebt gemaakt.

```
SELECT      a.voornaam, a.achternaam, b.titel, count(exnr)
FROM        auteurs a, boeken b, exemplaren e
WHERE       a.auteurnr = b.auteurnr
        AND e.boeknr = b.boeknr
GROUP BY    a.voornaam, a.achternaam, b.titel
HAVING      COUNT(exnr) >= 2
```

Over GROUP BY en HAVING is een powerpoint-presentatie beschikbaar

➔ Opdrachten

38. Uit welke plaatsen komen minstens tien leerlingen?
39. Van welke auteurs is het aantal aanwezige boeken groter dan vijf?
40. Noem de boeken die meer dan tien keer werden uitgeleend

10 NOT EXISTS

Hiervoor heb je gezien hoe je gegevens uit verscheidene tabellen kunt koppelen. Zo kun je gegevens over leerlingen en de boeken die deze leerlingen geleend hebben bij elkaar brengen. Daarvoor moest je een join maken tussen de tabellen Leerlingen, Uitleningen. De leerlingnummers brachten de juiste gegevens bij elkaar. Soms wil je nagaan of iets juist niet voorkomt

De schoolleiding wil het lezen op school bevorderen en start daarvoor een campagne. Ze wil de campagne richten op leerlingen die weinig lezen. De leerlingen die nog nooit een boek hebben geleend, behoren tot de doelgroep.
Hoe selecteer je deze leerlingen? Een koppeling maken helpt niet, dan krijg je juist gegevens over leerlingen die wel boeken hebben geleend. Je wilt juist een lijst van leerlingen die daarin niet voorkomen. Je kunt dit oplossen door het gebruik van het sleutelwoord NOT EXISTS.
De gevraagde leerlinggegevens vind je door een query met een speciale subquery te maken:

```
SELECT      *
FROM        leerlingen
WHERE       NOT EXISTS
            (SELECT     *
             FROM       uitleningen
            WHERE       uitleningen.llnr = leerlingen.llnr
            )
```

In deze query zitten twee nieuwe dingen. Ten eerste het sleutelwoord NOT EXISTS, 'bestaat niet'. Met dit sleutelwoord kun je zoeken naar dingen die **niet** voorkomen in de database. De tweede nieuwigheid zit in de WHERE-regel van de subquery. Daar wordt ook de kolom LLNR uit de tabel Leerlingen genoemd, terwijl de leerlingentabel verder niet in de subquery voorkomt. Met deze regel leg je een verband tussen de

hoofdquery en de subquery, het databasesysteem doet het volgende: in de hoofdquery wordt de leerlingentabel doorzocht. Daarvoor worden alle rijen afgelopen. Bij elke rij neemt het systeem het leerlingnummer en kijkt in de uitleningentabel of er rijen zijn met hetzelfde leerlingnummer. Als dat niet zo is (NOT EXISTS), wordt de leerling geselecteerd. Zo worden dus precies de leerlingen geselecteerd die geen boeken hebben geleend.

➔ Opdrachten

41. Welke boeken (auteurs, titels) zijn nog nooit uitgeleend?
42. Welke leerlingen (namen!) hebben nog nooit Nederlandse literatuur geleend?

11 VIEWS

Vaak is het handig om het resultaat van een query te bewaren. Zo kent het informatiesysteem wel tabellen met auteurs, boeken, exemplaren, maar een complete catalogus zit er niet in. Als je die zou willen raadplegen, moet je een ingewikkelde query bedenken en goed invoeren. Gelukkig kan dat veel eenvoudiger. Je maakt daarvoor een soort tabel die niet echt bestaat maar die je wel kunt raadplegen. Dat komt omdat het resultaat van een query, dat immers als een tabel aan ons wordt gepresenteerd, opgeslagen wordt alsof het een tabel is. Zo'n virtuele tabel noemen we een **view**. In standaard-SQL kun je zo een view maken.

```
CREATE VIEW     catalogus (voornaam, achternaam, titel, aantal_exempl)
        AS
SELECT          a.voornaam, a.achternaam, b.titel, count(exnr)
FROM            auteurs a, boeken b, exemplaren e
WHERE           a.auteurnr = b.auteurnr
      AND       e.boeknr = b.boeknr
GROUP BY        a.voornaam, a.achternaam, b.titel
```

Het resultaat van deze query is een tabel met de kolommen voornaam, achternaam, titel en het aantal exemplaren dat er van dat boek is. Deze tabel wordt op de een of andere manier opgeslagen en is voortaan te raadplegen alsof het een doodgewone tabel is. Het voordeel is dat ingewikkelde vragen hiermee een stuk eenvoudiger zijn te beantwoorden. Je doet dit door query's te maken die gegevens uit de view opvragen:

```
SELECT      …….
FROM        catalogus
WHERE       ….
```

➔ Opdrachten

43. Maak een view waarin is bijgehouden hoeveel boetes de leners hebben betaald. Per leerling moet hier het totaal van boetes voor die leerling staan
44. Welke leerlingen hebben meer dan € 25,00 aan boetes betaald? Gebruik de view van 43 bij het beantwoorden van deze vraag
45. De bibliothecaris wil eenvoudig kunnen zien welke boeken thuis, uitgeleend of in reparatie zijn. Help hem.

Hiervoor is eigenlijk **gebruikersinvoer** in een query nodig. Die maak je in Access heel simpel door bijv. in een query de regel 'Where boeknr = '66' te vervangen door **'Where boeknr = [Geef_boeknummer].** Access komt dan met een invoervenstertje waarin je het boeknummer kunt opgeven. Helemaal mooi is het als je ook nog met jokertekens werkt. Dan kun je elke invoer accepteren: **'Where naam like '*' & [geef_naam] & '***

Overzicht SQL-query
De basisstructuur van een query is:

```
select      <kolommen en/of functies>
from        <tabellen>
[ where     <voorwaarden> ]
[ order by  <sorteer-kolommen> ]
[ group by  <groepeer-kolommen> [ having <groep-voorwaarden> ] ]
```

De eerste twee regels zijn verplicht. De andere hoeven niet voor te komen, dit wordt aangegeven door de blokhaken.

SELECT	Achter SELECT geef je op welke kolommen de resultaattabel van de query moet hebben.
Kolommen	Je duidt kolommen aan met hun naam. Kolomnamen worden gescheiden door komma's. Wanneer er verwarring mogelijk is doordat er kolommen met dezelfde naam voorkomen (koppelingen), geef je kolommen aan met <tabelnaam>.<kolomnaam> (punt er tussen).
Alle kolommen	Alle kolommen uit een tabel krijg je met * of <tabelnaam>.*
Functies	Je kunt functies over een hele kolom laten uitrekenen: aantal rijen: COUNT(*), totaal: SUM(<kolomnaam>), gemiddelde: AVG(<kolomnaam>), maximum: MAX(<kolomnaam>), minimum: MIN(<kolomnaam>). Meestal selecteer je of functies, of kolommen, niet door elkaar. Uitzondering is het groeperen, zie daar.
DISTINCT	Door dit sleutelwoord direct achter SELECT te zetten, zullen er in het resultaat geen dubbele rijen verschijnen.
FROM	Hier geef je de tabel of tabellen op waaruit de gebruikte gegevens gehaald moeten worden. Tussen de tabelnamen moeten komma's.
Pseudoniemen	Wanneer je direct achter een tabelnaam een afkorting van de tabelnaam zet, kun je die afkorting in de rest van de query gebruiken in plaats van de tabelnaam zelf.
Koppelingen	Wanneer je meer dan één tabel gebruikt, maakt het systeem een koppeling tussen de tabellen. Dit betekent dat alle mogelijke combinaties van rijen gemaakt zullen worden. Meestal is dit niet gewenst, bij WHERE kun je dan door middel van een koppelvoorwaarde aangeven welke combinaties gemaakt moeten worden.
WHERE	Achter WHERE geef je met behulp van een voorwaarde aan welke rijen uit de brontabellen geselecteerd moeten worden.
Koppelvoorwaarden	Sommige voorwaarden dienen om aan te geven welke combinaties bij een koppeling gemaakt moeten worden. Meestal bestaat zo'n voorwaarde uit het gelijkstellen van kolommen die in beide tabellen voorkomen (bijvoorbeeld: leerlingen.llnr = uitleningen.llnr). AND, OR, NOT Afzonderlijke voorwaarden verbind je met AND of OR. In een reeks van AND en OR wordt de AND het eerst gedaan. Gebruik waar nodig haakjes. Met NOT kun je een voorwaarde omkeren.
Constanten	Zet enkele aanhalingstekens '...' om tekstconstanten of datums. NB Access gebruikt een apart datatype voor datums Getallen kun je zonder meer invoeren.
Operatoren	Gebruik =, <, >, >=, <= of o om te vergelijken. De < en > duiden bij tekst op alfabetische volgorde: < betekent dan 'eerder in het

alfabet'.

LIKE	Gebruik LIKE in plaats van = als je niet exacte gelijkheid bedoelt. In de tekstconstante kun je dan _ (Access:?) gebruiken voor één onbekende letter, % (Access:*) voor willekeurig veel letters. Voorbeeld: NAAM LIKE '_ob%' selecteert 'Bob', 'Rob' en 'Robert', maar niet 'Jacob'.
IS NULL	Gebruik IS NULL om te testen of er in een veld iets is ingevuld.
Subquery, IN	Binnen een query kun je een subquery maken voor een tussenresultaat. De voorwaarde is dan: <kolom> IN (<subquery>). Hiermee kun je nagaan of de gegevens in <kolom> voorkomen in de tabel die door de subquery wordt opgeleverd. De subquery moet een tabel met één kolom opleveren. Als je zeker weet dat de subquery maar één rij oplevert, bijvoorbeeld doordat er een functie als MAX gebruikt wordt, kun je in plaats van IN ook = gebruiken.
Subquery, EXISTS	Hiermee kun je nagaan of dingen wel of niet voorkomen. De voorwaarde is: [NOT] EXISTS (<subquery>). De subquery kan een willekeurige query zijn.
ORDER BY	Achter ORDER BY kunnen één of meer kolommen worden opgegeven, door middel van de kolomnaam of het rangnummer van de bij SELECT opgegeven kolommen. Er wordt gesorteerd op de eerstgenoemde kolom, bij gelijkheid op de tweede enzovoort. Na elke kolom kun je ASC (oplopend) of DESC (aflopend) opgeven. Standaard is oplopend.
GROUP BY	Met GROUP BY kun je de rijen van een tabel indelen in groepen die dezelfde gegevens in een kolom (of combinatie van kolommen) hebben. Die kolom(men) geef je achter GROUP BY op.
Functies	Bij SELECT zet je dezelfde kolommen als achter GROUP BY, plus functies voor het aantal, totalen en dergelijke. Deze functies worden dan voor elke groep apart uitgerekend.
HAVING	Bij HAVING kun je voorwaarden opgegeven die voor de groepen moeten gelden. Er is een belangrijk verschil met de voorwaarden die je bij WHERE opgeeft. Met de voorwaarden bij WHERE worden rijen geselecteerd voordat er gegroepeerd wordt. De voorwaarden bij HAVING gelden voor de groepen en de functies die hierin worden gebruikt. Groepen met meer dan tien rijen selecteer je bijvoorbeeld met HAVING COUNT(*) > 10.

➔ Opdrachten

Dit zijn de afsluitende opdrachten voor dit deel van de lesbrief. Het gaat over alle vaardigheden door elkaar. Je moet uitgaan van de situatie van 3 maart 2006.

46. Maak een overzicht - compleet met naam van auteur, titel, naam van leerling en klas - van alle openstaande uitleningen van leerlingen uit de zesde klas..
47. Maak een overzicht - compleet met naam van auteur, titel, naam van leerling en klas - van alle openstaande reserveringen
48. (Lastig!) Soms gaat er wel eens iets mis bij het maken van reserveringen. Er wordt dan een reservering voor een boek gemaakt terwijl er gewoon een uitleenbaar exemplaar aanwezig is. Schrijf een query die voor alle openstaande reserveringen nagaat of inderdaad alle exemplaren van dat boek uitgeleend zijn. Het is de bedoeling dat de query een uitleenbaar boek bij een reservering vindt, als dat er is.

Wanneer alles klopt, de boeken zijn inderdaad uitgeleend, dan vindt de query niets (geen fouten).

49. Peter Icke komt in de bibliotheek met de vraag of hij het boek *Giph* van Ronald Giphart kan lenen. Ga na of er uitleenbare exemplaren zijn die aan hem geleend kunnen worden.
50. Maak deze query geschikt voor elk boek, d.m.v. gebruikersinvoer in combinatie met jokertekens. Nog mooier is als je laat zoeken op auteursnaam en/of titel.
51. Wanneer een gereserveerd boek beschikbaar komt, krijgt de betreffende leerling bericht en kan hij of zij het boek binnen vijf dagen ophalen. Als dat niet gebeurt, vervalt de reservering.
    Schrijf een query die een lijst maakt van reserveringen die op 3 maart 2006 vervallen. Zet ook de titel van het boek en de schrijver in het resultaat.
    NB het is nu 3 maart 2006! Je moet dus terugrekenen! In de tabel Uitleningen kun je zien of een gereserveerd boek is opgehaald!
52. In de schoolbibliotheek geldt een uitleentermijn van twee weken.
    Maak een lijst van alle uitgeleende boeken waarvan de uitleentermijn verstreken is (dus uitgeleend vóór ….). Als een boek na drie weken nog niet terug is, krijgt de lener een aanmaning.
53. Maak een lijst van aanmaningen die op 3 maart 2006 (dus vandaag!) verstuurd moeten worden. In het resultaat moeten ook de naam van die leerling, de titel van het boek en de schrijver staan, zodat de aanmaning direct afgedrukt kan worden.
54. Een leerling brengt *Figuranten* van Arnon Grunberg terug. Op de kaft staat dat dit boek het nummer 66 heeft. Het systeem moet nu nagaan of er soms een reservering voor dit boek is.
    Schrijf een query die nagaat of er een reservering is en zo ja, welke leerling bericht moet krijgen dat hij of zij het boek kan ophalen. In het resultaat moeten ook de naam van die leerling, de titel van het boek en de schrijver staan, zodat het afhaalbericht direct afgedrukt kan worden.
    Oude reserveringen, waarvoor al eerder een exemplaar beschikbaar was, moet je natuurlijk niet meetellen. Het kan wel zijn dat er twee of meer reserveringen openstaan. In dat geval gaat het boek naar de leerling die het langst geleden de reservering gemaakt heeft.
55. Om te beslissen of er van sommige boeken extra exemplaren aangeschaft moeten worden, wil de schoolleiding weten hoe vaak boeken zijn uitgeleend. Maak een overzicht van boeken, met titel en auteur, waarin per boek het aantal uitleenbare exemplaren staat.
56. Maak een overzicht van boeken, met titel en auteur, waarin per boek is aangegeven hoe vaak het is uitgeleend na 1 september 2004.

12 HET RELATIONELE MODEL EN DE STRUCTUUR VAN EEN GEGEVENSBANK

In een relationele database worden de gegevens opgeslagen in tabellen. Tabellen bestaan uit rijen en kolommen. Wanneer je aangeeft welke tabellen en kolommen er in een database zitten, is de beschrijving van de database nog niet compleet. In het relationele model worden voor elke database ook beperkingen en verbanden tussen tabellen onderling vastgelegd. Verder kun je aangeven of gegevens altijd ingevuld moeten worden of dat een plaats 'open' mag blijven.
We zullen deze begrippen nu eerst uitwerken. Daarna bekijken we hoe je een database definieert in SQL.

Sleutels en verwijzingen

!
Elke tabel heeft een kolom of een combinatie van kolommen die uniek is. Dat wil zeggen dat (een combinatie van) gegevens in die kolommen maar één keer in een rij mogen voorkomen. Dit is nodig om de rijen uit elkaar te houden.
Die kolom of combinatie van kolommen noemen we de sleutel van de tabel, of ook

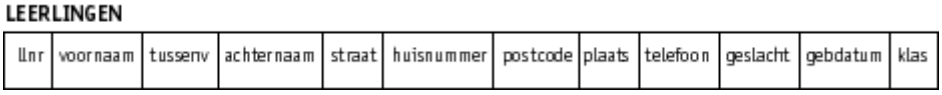
! wel de **primaire sleutel** van de tabel, in het Engels 'primary key'.Voor een primaire sleutel geldt:
- In elke rij moeten in de sleutelkolom(men) waarden zijn ingevuld.
- Deze waarden (of combinaties van waarden) zijn verschillend voor alle rijen.

In de bibliotheekdatabase hebben alle boeken in de tabel Boeken een boeknummer. Alle boeken hebben verschillende nummers. *Max Havelaar* van Multatuli heeft bijvoorbeeld het boeknummer 43 en er is geen ander boek met het nummer 43. Een boeknummer kan maar één keer voorkomen, het boeknummer is uniek. Boeknummer is de primaire sleutel van deze tabel.
De tabel Exemplaren heeft een sleutel die uit twee kolommen bestaat: het boeknummer plus het exemplaarnummer. Van eenzelfde boek kunnen meer exemplaren zijn, het boeknummer is daarom niet uniek. Er is echter maar één eerste exemplaar van een boek, één tweede exemplaar enzovoort. Van de *Max Havelaar* zijn twee exemplaren, in de tabel staan daarom twee rijen met boeknummer 43, één met het exemplaarnummer 1 en één met het exemplaarnummer 2. De combinatie van boek- en exemplaarnummer is daarom wel uniek. Om een exemplaar aan te duiden moet je dus zowel het boeknummer als het exemplaarnummer gebruiken.
In deze twee voorbeelden zie je dat er vaak nummers gebruikt worden om dingen (boeken, exemplaren, leerlingen) uniek aan te duiden. Meestal kan dat ook met een combinatie van gegevens. Er is bijvoorbeeld maar één *Max Havelaar* van Multatuli. Bij leerlingen is het al lastiger.
De tabel Boeken heeft een primaire sleutel, het boeknummer. Max Havelaar van Multatuli heeft bijvoorbeeld het boeknummer 43. Bij de combinatie van titel en auteurnummer zullen ook geen dubbele voorkomen. Een auteur zal nooit twee verschillende boeken met dezelfde titel schrijven. De combinatie titel plus auteur is daarom een mogelijke sleutel.
Sleutels zorgen dus voor een unieke identificatie: er is maar één leerling met het leerlingnummer 61. Dit is van belang als je in andere tabellen informatie over leerlingen op wilt slaan.
In de tabel Uitleningen wordt vastgelegd welke boeken leerlingen geleend hebben. We geven dit aan door het leerlingnummer van de betreffende leerling op te nemen. Dit nummer is uniek, we weten dus precies welke leerling het boek geleend heeft. Met dat leerlingnummer wordt een verband gelegd tussen de tabellen Uitleningen en Leerlingen.
Er is ook een verband tussen de tabellen Uitleningen en Exemplaren. De laatste tabel heeft een sleutel die bestaat uit twee kolommen: boeknr en exnr. In de tabel Uitleningen nemen we die allebei op, zodat we precies weten welk exemplaar is uitgeleend.
! Zo'n verband tussen de gegevens in de ene tabel en de gegevens in een andere tabel heet een **verwijzing**, of ook wel **referentiesleutel**. De Engelse term is foreign key.
Verwijzingen kunnen door een relationeel databasesysteem ten dele gecontroleerd worden. Stel, je wilt een boek uitlenen aan een leerling met leerlingnummer 53. Dat wordt in het systeem vastgelegd met een nieuwe rij in de tabel Uitleningen. Het systeem kan dan nagaan of er wel een leerling met het leerlingnummer 53 in de tabel Leerlingen voorkomt. Hierdoor is het onmogelijk boeken uit te lenen aan leerlingen die (volgens het systeem) helemaal niet bestaan, of boeken uit te lenen die (alweer volgens het systeem) helemaal niet in de bibliotheek zijn.
! Het controleren van de verwijzingen noemen we het bewaken van de **referentiële integriteit**, in het Engels: referential integrity.

Waarden
Je kunt ook eisen opleggen aan de waarden die in een kolom worden ingevuld .Veel van die eisen kunnen door het databasesysteem gecontroleerd worden. Er zijn drie soorten regels voor de waarden in een kolom:

1 Je kunt eisen dat een waarde verplicht wordt ingevuld, voor elke rij moet in de genoemde kolom een waarde worden ingevuld. Als je niets invult weigert het systeem die rij in een tabel op te nemen. Dit heet een **niet-leeg**-regel (**not-null**-regel). Het Engelse woord 'null' betekent hier 'zonder waarde' ofwel 'niet ingevuld'. Dit is iets anders dan het getal nul of een stel spaties, dan is er immers wel degelijk iets ingevuld.
2 Je kunt eisen dat er waarden van een bepaald soort worden ingevuld: tekst, getallen, datums of andere soorten gegevens. De mogelijke soorten zijn vaak afhankelijk van het systeem. MySql en Access hebben verschillende mogelijkheden.
3 Voor sommige gegevens gelden heel specifieke eisen. Bij het geslacht van een leerling bijvoorbeeld zijn de mogelijke waarden alleen 'm' of ‘v’. De meeste databasesystemen hebben mogelijkheden om dergelijke regels aan te geven, maar ze verschillen sterk. We gaan er daarom niet verder op in.

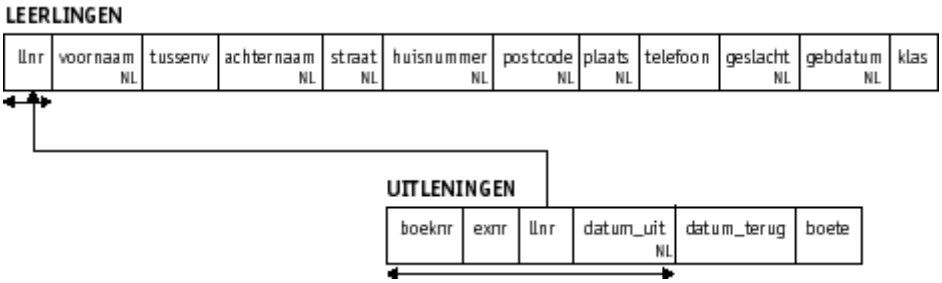
Het strokendiagram
Voordat je gegevens in een informatiesysteem kunt invoeren moet er een tabellenstructuur zijn. Dat is een overzicht van de tabellen compleet met kolommen, sleutels en verwijzingen. Je kunt het vergelijken met het ontwerp van een blanco invulkaart. Je definieert wel de in te vullen velden maar zet er nog niets in.
Voor het noteren van het ontwerp van een database wordt vaak een strokendiagram gebruikt. Dit is een schema waarin de tabellen als stroken verschijnen met daarin de namen van de kolommen. In het diagram kun je behalve de tabellen en de kolommen ook de primaire sleutels, verwijzingen en niet-leeg-regels aangeven. Een tabel geef je aan door een strook met daarin de kolomnamen.



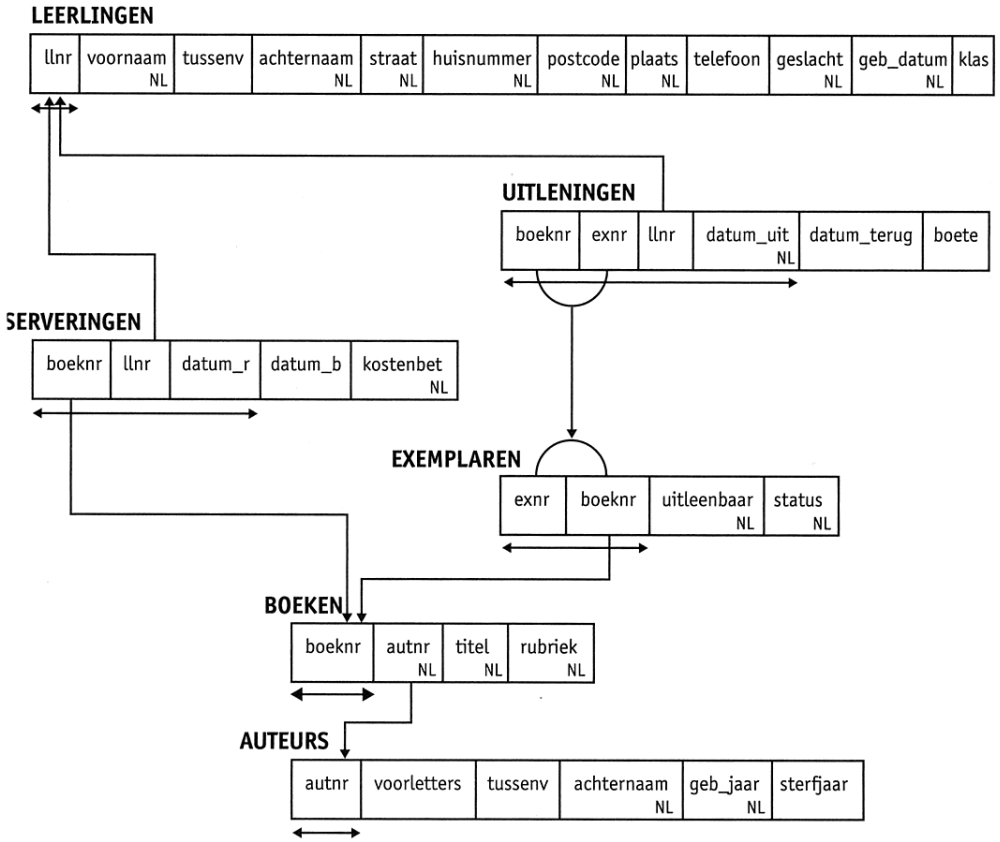
De niet-leeg-regels geef je aan door bij de betreffende kolommen ‘NL’ te schrijven:

De primaire sleutel wordt aangeduid met een dubbele pijl onder de kolommen die deel uitmaken van de sleutel. Bij de sleutelkolommen mag je de NL-aanduiding weglaten, omdat bij een sleutel altijd een waarde moet worden ingevuld.

Een verwijzing wordt aangeduid met een pijl van de ene tabel naar de andere. De pijlpunt wijst naar gegevens waarnaar verwezen wordt. Meestal is dit een sleutel.



Het complete strokendiagram voor de bibliotheekdatabase ziet er dan zo uit:



➔ **Opdracht**

57. Tussen de tabel Uitleningen en de tabel Exemplaren zit een ingewikkelde verwijzing. Omschrijf nauwkeurig wat de betekenis van deze verwijzing is.

13 HET BOUWEN EN VERANDEREN VAN EEN DATABASE IN SQL

Het strokendiagram van een database laat zich direct vertalen in SQL-instructies voor het definiëren van een nieuwe database. Er moet echter nog één ding aan worden toegevoegd: voor elke kolom moeten we beslissen wat voor soort gegevens moeten worden ingevuld.

! We noemen dit het datatype. Standaard SQL kent een beperkt aantal datatypen. Een daarvan is CHAR, dit staat voor tekst. Wanneer je dit datatype gebruikt moet je het maximale aantal tekens opgeven. CHAR(15) staat voor een tekst van maximaal 15 tekens. Een soortgelijk datatype is VARCHAR, dit staat voor tekst met een variabele lengte. Met gegevens van deze typen kan niet gerekend worden. De tekens worden opgeslagen om te worden gelezen en meer niet. Natuurlijk kun je ze

sorteren en kun je kolommen combineren en aan elkaar plakken.
Wanneer er wel gerekend moet kunnen worden hebben we de keus uit een aantal datatypen. Gaat het om betrekkelijk kleine gehele getallen, dan is SMALLINT heel gebruikelijk. Voor grote gehele getallen gebruiken we het datatype INTEGER. Het verschil zit in de hoeveelheid geheugenplaatsen die voor de opslag ervan nodig zijn.
Een getal met cijfers achter de komma wordt aangeduid met DECIMAL, FLOAT of REAL. Bij DECIMAL kun je het totaal aantal cijfers opgeven en het aantal cijfers achter de komma. DECIMAL(8,2) staat voor getallen van 8 cijfers waarvan 2 achter de komma.

Om je een idee te geven:

TINYINT: -2^7 - $+2^7$
SMALLINT: -2^{15} - $+2^{15}$
INTEGER: -2^{31} - $+2^{31}$

Een datum is van het datatype DATE. Feitelijk wordt ook hier een getal opgeslagen, maar op een speciale manier. Het aantal dagen sinds een begindatum, vaak is dat 30 december 1899, wordt uitgerekend en kan op verschillende manieren worden weergegeven. TIME gebruik je voor het weergeven van tijdstippen. Deze datatypen zitten in het standaard-SQL van MySql. Voor elk systeem geldt dat je in de documentatie moet opzoeken wat mogelijk is. Er zijn namelijk nogal eens uitbreidingen buiten standaard SQL om. Als voorbeeld geven we hieronder aan hoe je de tabellen Auteurs en Boeken moet construeren in standaard SQL::

```
CREATE TABLE auteurs
(       auteurnr            smallint        NOT NULL,
        voornaam            char(12)        ,
        tussenvoegsel       char(7)         ,
        achternaam          char(20)        NOT NULL,
        geb jaar            integer         ,
        sterfjaar           integer         ,
        PRIMARY KEY (auteurnr)
);
```

Let op de komma's achter de regels; overal, behalve na de laatste!

```
CREATE TABLE boeken
(       boeknr              smallint        NOT NULL,
        auteurnr            smallint        NOT NULL,
        titel               char(20)        NOT NULL,
        rubriek             char(5)         NOT NULL,
        PRIMARY KEY (boeknr),
        FOREIGN KEY (auteurnr) REFERENCES auteurs(auteurnr)
);
```

In deze regels hierboven vind je tabellen, kolommen, datatypen, niet-leeg-regels, primaire sleutels en verwijzende sleutels (foreign keys).
Als de sleutel uit meer kolommen bestaat, komen die, gescheiden door komma's, tussen de haakjes achter PRIMARY KEY. Hetzelfde geldt voor de verwijzingen: ook hier kun je combinaties van kolommen opgeven.

Wijzigen van een database
Het vullen en veranderen van een database gebeurt bij ACCESS met een gebruikersvriendelijke interface. MySql gebruikt instructies van standaard SQL om de tabellen te vullen en te wijzigen. Er zijn drie belangrijke instructies: INSERT (rijen toevoegen), UPDATE (gegevens veranderen) en DELETE (rijen weggooien).
Het invoegen van rijen in een tabel gaat met INSERT:

```
INSERT INTO boeken VALUES (186,47, 'De donkere kamer van Damocles',
'nederlands')
```

Hiermee wordt een enkele rij in de tabel gevuld. Voordat de rij daadwerkelijk wordt opgenomen controleert SQL of aan alle voorwaarden is voldaan. Dat gaat dan vooral om datatypen, NULL-waarden en integriteit: geen dubbele waarden bij sleutels en bij verwijzingen geen waarden toelaten als die niet in de andere tabel voorkomen.Voor elke volgende rij moet dit worden herhaald. Je ziet dat in de SMALLINT-kolommen getallen zonder aanhalingstekens worden genoteerd, maar de overige kolommen hebben wel aanhalingstekens. Dat geldt trouwens ook voor datumkolommen.

! Als bij het vullen niet alle kolommen gevuld worden, of als dat niet gebeurt in de volgorde van de tabel, moet de naam van iedere kolom worden vermeld, in de juiste volgorde.

```
INSERT INTO auteurs (auteurnr, naam) VALUES (421. 'Toonder')
```

Het is ook mogelijk om rijen te wijzigen of aan te vullen.

```
UPDATE      auteurs
SET         voornaam ='Marten'
            geb_jaar = 1912
            sterf_jaar = NULL
WHERE       auteurnr = 421
```

Je ziet in deze instructie ook een regel met WHERE. Hiermee selecteer je de rijen waarop de wijziging moet worden uitgevoerd. Dit mag er meer dan één tegelijk zijn, je kunt dus een serie wijzigingen in één keer doorvoeren. Het gebruik van WHERE is hetzelfde als bij query's.
Als het met het construeren van een tabel is misgegaan, kan de hele tabel weggegooid worden met:

```
DROP TABLE auteurs
```

Maar nu is ook de hele tabelstructuur verdwenen. Als je één of meer rijen uit een tabel wilt verwijderen, is daar een ander opdracht voor. Bijvoorbeeld als een boek afgeschreven wordt.

```
DELETE
FROM        boeken
WHERE       boeknr = 184
```

Als je de tabel wilt leegmaken maar niet de tabelstructuur wilt weggooien, kun je de voorwaarde die begint met WHERE gewoon weglaten. Nu wordt de hele tabel leeggemaakt. De tabel bestaat nog wel, maar er zit geen enkele rij meer in.

➔ Opdrachten

Deze opdrachten moet je op papier maken! Alleen als ze op papier zijn 'goedgekeurd', mag je ze je echt uitvoeren

58. Op 3 maart 2006 leent Aniel Mangakas (llnr 83) exemplaar 1 van '*A Catcher in the Rye*' (boeknr 31). Schrijf de SQL-instructie om dit in de bibliotheekdatabase vast te leggen
59. Rafael Menendez is verhuisd naar Veldwachterserf 14 in Houten, postcode 3991 KW. Schrijf de SQL-instructie om dit in de bibliotheekdatabase vast te leggen
60. De reservering die Astrid Poolster (llnr 78) op 27 februari 2006 gemaakt zou

hebben voor boek 71 blijkt een administratieve fout te zijn, Schrijf de SQL-instructie om de reservering uit de bibliotheekdatabase te verwijderen.

61. Hieronder zie je het strokendiagram voor een sportclub. Er zijn tabellen voor de gegevens over leden, teams en het wedstrijdprogramma van de teams. Alle andere zaken, zoals uitslagen van wedstrijden, worden in dit voorbeeld niet bekeken.
62. Schrijf de SQ:-regels die je nodig hebt om deze database te creëren. Kies zelf geschikte datatypen voor de kolommen. Denk aan de primaire sleutels en de verwijzingen.