- 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 zelfs 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.

10.3 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>
```

We zouden om te beginnen eens kunnen kijken wat er allemaal voor 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
```

SELECT
FROM

Achter SELECT geef je de kolommen op die je in de uitvoer wilt zien, gescheiden door komma's. Achter FROM geef je de naam van de tabel (of tabellen) waaruit deze kolommen komen.
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 achterdeurtje om deze SQL-query wat eenvoudiger te maken. Als je alle kolommen wilt hebben, dan is het voldoende als je een sterretje invult. 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.

Figuur 5
De leerlingentabel.

LEERLINGEN : Tabel

LLNR	VOORNAA	TUSSEI	ACHTERNAAM	STRAAT	HUISN	POSTCODE	PLAATS	TELEFOON	GESLAC	GEB_DATUM	KLAS
51	Janco		Dijke van	Houtenseweg	8	3984LJ	Odijk	03405-782	m	21-12-83	5a
52	Rene		Bokker	Graaf Boudewijnlaan	35	6464DV	Nieuwegein	06402-140199	m	26-05-84	5b
53	Marco	van	Bemmel	Wielingenstraat	6	3522PG	Utrecht	030-1122337	m	10-01-83	4a
54	Petra		Rohstein	Vlierweg	40	3991BD	Houten		v	04-05-83	4b
55	Nourdine		Abloukie	Potgieterstraat	25 bis	3532VP	Utrecht	030-864548	v	28-12-83	5a
66	Nathalie	van	Horp	Auriollaan	71	3527ES	Utrecht	030-231231	v	17-06-85	6b
57	Debby	van der	Boogaard	Ferdinand Bolstraat	70	3584AS	Utrecht	030-456132	v	24-07-84	5b
58	Monique		Aalbers	Lange Uitweg	18	3998WE	Schalkwijk	03403-35676	v	24-09-85	6b
59	Yvanka		Uittenbogaard	Huize de Geerlaan	121	3523JN	Utrecht		v	27-04-85	6a
60	Antoinette		Bosveld	Boslaan	16	3981XD	Bunnik	03404-54654	v	06-11-85	6b
61	Danielle		Hoevelaken	Frederik van Eedenstr.	26	3451AZ	Vleuten	03401-6234	v	27-01-84	5a
62	Arnold	de	Jong	Oudegeinlaan	8	3525KK	Utrecht	030-543632	m	13-07-83	4b
63	Patricia		Bloesem	Albatrosstraat	30	3582EX	Utrecht	030-768431	v	09-12-83	5a
64	Robert		Okkernoot	Herenweg	12	3991DS	Houten	03403-35687	m	31-03-83	4a
66	Dennis		Smets	Merelhaag	30	3993XA	Houten	03403-93211	m	15-10-84	5b
67	Aroena		Makassar	Boslaan	38	3981XD	Bunnik		v	23-05-85	6b

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 exact opgeven welke kolommen je wilt hebben. Ook de volgorde waarin je de kolommen vermeldt, is van belang; de kolommen komen precies in die volgorde op het scherm.

```
SELECT    VOORNAAM, TUSSENVOEGSEL, ACHTERNAAM, KLAS
FROM      LEERLINGEN
```

Sorteren

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

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 ten slotte 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

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. Dat wordt in paragraaf 6 besproken.

Ascending
Descending

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.

DISTINCT

```
SELECT    DISTINCT PLAATS
FROM      LEERLINGEN
```

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

10.4 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'
```

ERE

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. Zo kun je ook een lijst van alle meisjes laten maken.

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 voorzichtig 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.

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 dat ongeveer klopt, en dan zien we zelf eventueel wel of we de vraag nog moeten verfijnen.

LIKE

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 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 _ 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 'Frederi_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.

erator

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 ^=, != of ≠.

10.5 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.

AND

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.

NULL

In SQL betkent 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'
```

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 note-

ren we de query zo dat hij overzichtelijk en leesbaar is. Elk sleutelwoord op een nieuwe regel.

OR

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 4a 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 eruitziet 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 moest 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, POST
          CODE, 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.

10.6 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 willen 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 kwijt 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. Het moet natuurlijk wel een kolom met getallen zijn. 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 dát 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.

ROUP BY

Om te groeperen heeft SQL de opdracht GROUP BY. De query wordt dan:

```
SELECT    KLAS, COUNT(*)
FROM      LEERLING
GROUP BY  KLAS
```

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>.

10.7 Een query binnen een query

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 vervolgens van wie dat nummer is.

```
SELECT    LLNR
FROM      UITLENING
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      UITLENING
WHERE     BOETE =
          (   SELECT    MAX(BOETE)
              FROM      UITLENING
          )
```


In plaats van het bedrag € 4,00 is de hele query ingevuld die dat bedrag heeft opgeleverd. We spreken hier van 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      LEERLING
WHERE     LLNR =
          (   SELECT    LLNR
              FROM      UITLENINGEN
              WHERE     BOETE =
                        (   SELECTMAX(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.

IN

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     LEERLING
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. Die staat in paragraaf 10.2. 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.

10.8 Combinaties van tabellen

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:TABEL JONGENS	
NAAM	WOONPLAATS
Jan	Leiden
Piet	Assen
Kees	Arnhem

Tabel:TABEL MEISJES	
NAAM	WOONPLAATS
Inge	Leiden
Marleen	Assen
Joyce	Arnhem

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

```
SELECT   *
FROM..   JONGENS, MEISJES
```

ppeling van
ellen

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.

Maar gelukkig is daar wel een en ander aan te doen. In de eerste plaats heb je 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. In de tweede plaats werken BDMS'en met indexen, waarmee je snel in tabellen kunt zoeken. In hoofdstuk 11 komen we terug op het begrip index.

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

Er valt nog wel wat te sleutelen aan de query. In de eerste plaats hoeft die woonplaats niet twee keer vermeld te worden. En dan is er nog een trucje. Die kolomnamen zijn soms wat aan de lange kant. Als ze in een query zo'n acht of meer keer voorkomen krijg je het gevoel dat dat wel een beetje veel van het goede is. Daarom kun je in de query een afkorting voor de tabelnamen opnemen. Heel simpel, je zet er gewoon een letter achter.

udoniem

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.

```
SELECT    VOORNAAM, TUSSENVOEGSEL, ACHTERNAAM, DATUM_UIT, DATUM_TERUG, BOETE
FROM..    LEERLINGEN, UITLENINGEN
```

Zonder nadere voorwaarden krijgen we dus een combinatietabel waarin elke rij uit de tabel Leerlingen gecombineerd wordt met elke rij uit de tabel Uitleningen. Dat is natuurlijk niet de bedoeling. We hebben niets aan een combinatieregel waar gegevens van de ene leerling gecombineerd worden met de uitlening van iemand anders. Het moet wel om dezelfde leerling gaan. Het verband tussen de twee tabellen zit in de kolommen LInr. In de tabel Uitleningen zit ook een kolom LInr. 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
```

Zonder die voorwaarde heb je niets aan de combinatietabel. Let dus op de voorwaarde die je stelt en kijk daarvoor goed naar de tabellenstructuur. 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.

10.9 Uitbreidingen

In deze paragraaf bespreken we nog enkele mogelijkheden van SQL. Deze elementen van SQL zijn lastiger dan hetgeen we tot nu toe hebben behandeld, maar je kunt er bijzondere vragen mee oplossen.

Groeperen met voorwaarden

Van sommige boeken zijn er verschillende exemplaren en van andere maar een enkel exemplaar. We willen een lijst maken van alle boeken