Informatie uit één tabel

Leereenheid 4

Informatie uit één tabel

INTRODUCTIE

Tot nu toe zijn we vooral bezig geweest met de theorie van relationele databases, zowel in Inleiding informatica als in deze cursus. Deze leereenheid is de eerste van een serie praktische leereenheden, waarin we zien hoe de tabellen van een databaseontwerp worden gecreëerd (leereenheid 9), hoe de gegevens erin komen en hoe die gegevens beheerd kunnen worden (leereenheid 8), en vooral hoe we de informatie die we nodig hebben uit de database kunnen krijgen (leereenheden 4 tot en met 7).

Het statement om gegevens op te vragen (`select`) kan complexe vormen aannemen. In deze leereenheid houden we het betrekkelijk eenvoudig, door de bevragingen te beperken tot één tabel.

LEERDOELEN
Na het bestuderen van deze leereenheid wordt verwacht dat u
- in natuurlijke taal geformuleerde bevragingen die betrekking hebben op gegevens in één tabel kunt omzetten naar SQL-`select`-statements
- vertrouwd bent met het gebruik van de volgende SQL-elementen: projecties; de pseudo-operator `distinct`; de datatypen `integer`, `numeric`, `varchar`, `date`, `time` en `timestamp` met hun operatoren; impliciete en expliciete typecasting; functies; selecties en logische operatoren; verzamelingsoperatoren, in het bijzonder de vereniging; `order by`.

De studielast van deze leereenheid bedraagt 7 uur.

Studeeraanwijzing

We gaan ervan uit dat u de code in deze leereenheid uitprobeert in de meegeleverde SQL-omgeving, en dat u ermee experimenteert. Dat wil zeggen, kijk wat er gebeurt als u iets in de code verandert (een conditie weglaten of veranderen bijvoorbeeld) en verzin ook gerust uw eigen vragen en probeer daarop het juiste antwoord te krijgen.

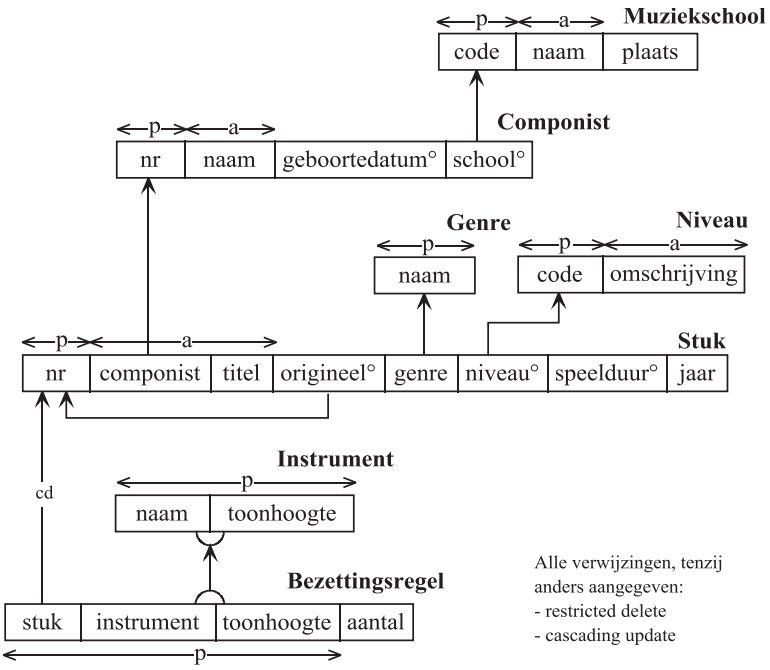
LEERKERN

1 De Muziekdatabase

We introduceren in deze leereenheid de voorbeelddatabase voor deze en de twee volgende leereenheden: de Muziekdatabase.

Beschrijving
De Muziekdatabase is in gebruik bij een overkoepelende organisatie van muziekscholen. Deze kunnen online een database van solo- en samenspeelstukken raadplegen.
Veel stukken zijn door docenten van de scholen ingebracht en door henzelf gecomponeerd, bewerkt of gearrangeerd. Bij niet-originele stukken (bewerkingen) kan informatie over de originele compositie worden opgezocht.
Elk stuk heeft een genre (bijvoorbeeld jazz) en een niveau (bijvoorbeeld A = eenvoudig). Er zijn geen eigenschappen van een originele compositie die door een bewerking worden 'geërfd'. Zelfs titel en genre mogen verschillen.
Sommige originele stukken zijn niet bedoeld als speelstuk, maar alleen opgenomen voor referentiedoeleinden. De echte *speelstukken* zijn de stukken waaraan een niveau is toegekend. Hiervan is altijd de instrumentbezetting opgenomen.
De maker van een stuk heet *componist*, ook als het om een bewerking gaat.

Strokendiagram
Figuur 4.1 toont het strokendiagram van de Muziekdatabase.



FIGUUR 4.1 Strokendiagram Muziekdatabase

Toelichting
We bespreken het diagram van boven naar beneden, dus van ouder naar kind.

– Een muziekschool wordt geïdentificeerd door een unieke code (primaire sleutel) en tevens door een unieke naam (alternatieve sleutel). Van elke muziekschool is de plaats(naam) vermeld. Men heeft het niet nodig gevonden deze te standaardiseren, vandaar dat een aparte tabel Plaats ontbreekt.
– Componisten worden op een vergelijkbare manier geïdentificeerd met een uniek nummer (nr). Ook zij hebben een unieke naam. Daarnaast kunnen een geboortedatum en de muziekschool waaraan zij verbonden zijn worden opgenomen.
– Stukken hebben een uniek stuknummer (primaire sleutel). De alternatieve sleutel over componist en titel impliceert dat stukken van één componist verschillende titels moeten hebben. De optionele kolom origineel bevat voor bewerkingen een (recursieve) verwijzing naar het origineel. Verder hoort elk stuk tot een genre en hebben de meeste stukken (de *speelstukken*) ook een niveau. Voor genres en niveaus zijn er standaardisatietabellen Genre en Niveau. Tot slot kan bij ieder stuk de speelduur worden opgenomen en heeft elk stuk een jaartal.
– De tabel Bezettingsregel geeft voor elk speelstuk de bezetting. Ziet men een muziekstuk als een muzikaal gerecht, dan zou men kunnen zeggen dat Bezettingsregel de ingrediënten bevat.
– Instrument bevat de gestandaardiseerde instrumenten.

De refererende-actieregels zijn op één uitzondering na default: restricted delete en cascading update. De uitzondering is de cascading delete voor de verwijzende sleutel van Bezettingsregel naar Stuk.

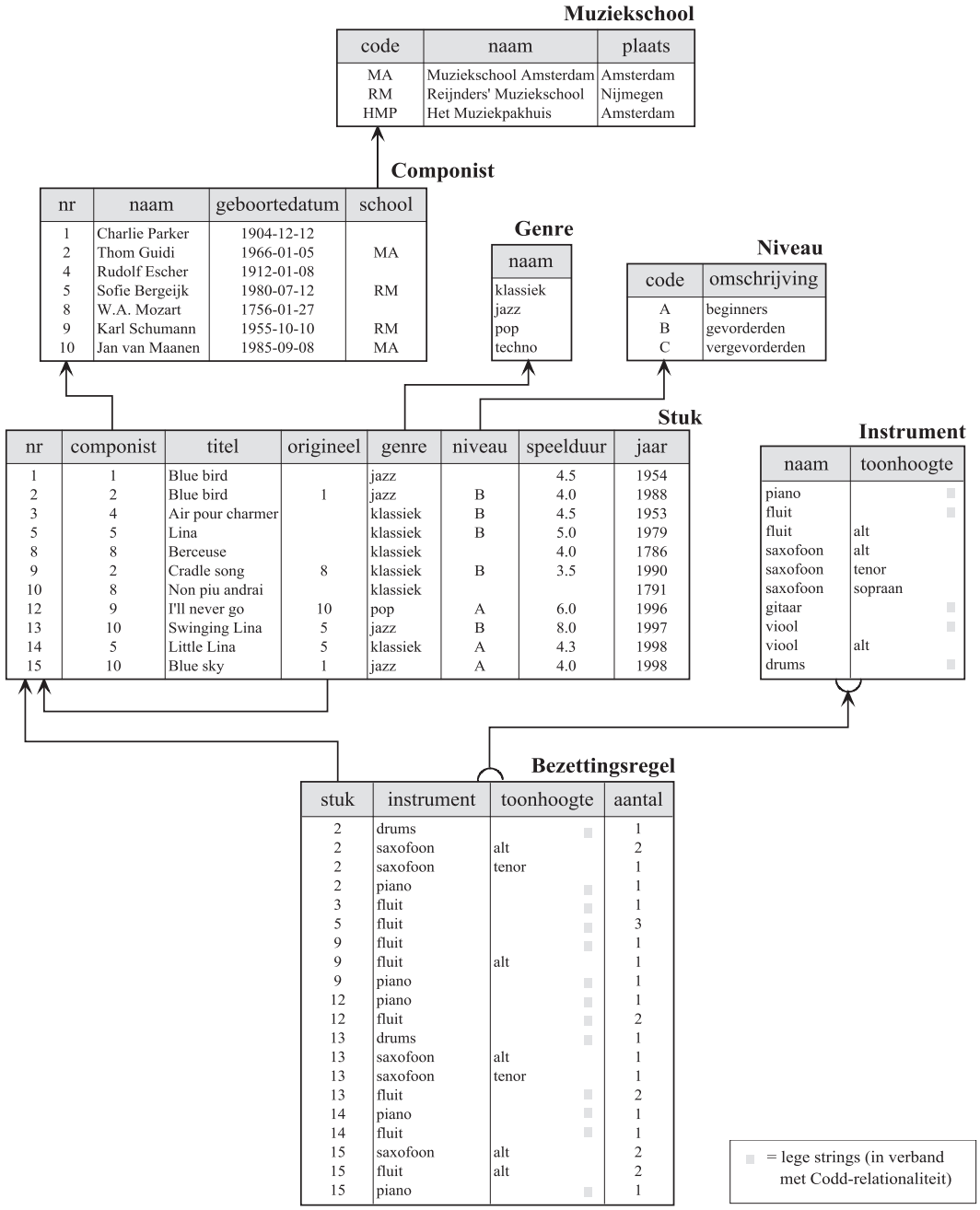
Het diagram van figuur 4.2 bevat een voorbeeldpopulatie. Deze vindt u ook terug in de bijlage achterin dit deel.

Opmerking
Stuk 10 heet eigenlijk 'Non più andrai', maar omdat accenten zich op verschillende systemen soms anders gedragen, laten we het accent maar weg.

Bijzondere beperkingsregels
Behalve de regels die grafisch kunnen worden weergegeven in de diagrammen, gelden voor de database ook nog bijzondere beperkingsregels. In voorbeeld 4.1 volgt zo'n regel.

VOORBEELD 4.1 (bijzondere beperkingsregel)

Een bewerking is altijd gemaakt van een origineel en dus nooit van een andere bewerking. Stuk 2 bijvoorbeeld is een bewerking van stuk 1. Dan moet stuk 1 een origineel zijn. Dat klopt: bij dat stuk is de kolomwaarde origineel niet ingevuld (null). Een speciaal geval van deze regel is dat een stuk geen bewerking mag zijn van zichzelf.
Er zijn verschillende manieren om zulke regels af te dwingen, onder meer via triggers. Deze worden behandeld in leereenheid 12.



FIGUUR 4.2 Voorbeeldpopulatie Muziekdatabase

De volgende opgave is bedoeld om wat thuis te raken in de structuur van de Muziekdatabase en van de voorbeeldpopulatie. Het is géén SQL-opgave.

OPGAVE 4.1

a Welke muziekschool heeft geen enkel stuk bijgedragen aan de database?

b Stukken met een niveau-aanduiding zijn bedoeld als speelstuk (voor samenspel of solo). Ga na dat van alle speelstukken de bezetting is opgegeven.

c Mogen we, vanuit de databasestructuur bekeken, spreken over *het* stuk met titel 'Non piu andrai'? Waarom (niet)?
d Is 'Swinging Lina' van Jan van Maanen een origineel of een bewerking?
e Eén componist heeft een door hemzelf gecomponeerd stuk bewerkt. Welke componist is dat en om welk stuk gaat het?
f Welke bewerkingen hebben een andere titel dan hun origineel?

2 Projecties: select ... from

Brontabel

Resultaattabel

Een `select`-statement is een operatie op een tabel-operand. De operand staat vermeld in de `from`-clausule en heet de *brontabel*. Het resultaat van elke `select`-query is weer een tabel: de *resultaattabel*. Een `select`-statement is daarom een relationele operatie.

De resultaattabel komt in een aantal stappen (via deeloperaties) tot stand. In deze paragraaf bekijken we enkele eenvoudige manieren om een resultaattabel af te leiden uit een brontabel.

Let op!

Een *projectie* ligt vast in de `select`-clausule. Dit kan verwarring geven met de *selectie*operatie, die we in paragraaf 6 behandelen.

2.1 PROJECTIE OP KOLOMVERZAMELING

Projectie

De meest eenvoudige `select`-operatie is de *projectie* op een deelverzameling van de verzameling kolommen. De gewenste kolommen worden opgesomd in een *kommalijst*:

```
select componist, titel, niveau
from   Stuk;
```

Resultaat:

```
COMPONIST TITEL            NIVEAU
========= ================ ======
        1 Blue bird        <null>
        2 Blue bird        B
        4 Air pour charmer B
        5 Lina             B
        8 Berceuse         <null>
        2 Cradle song      B
        8 Non piu andrai   <null>
        9 I'll never go    A
       10 Swinging Lina    B
        5 Little Lina      A
       10 Blue sky         A
```

Projectie op alle kolommen

Vaak willen we 'even snel' een tabelinhoud bekijken. Voor zo'n snelle ad-hoc-query kan een sterretje gebruikt worden als afkorting voor de volledige kolommenlijst:

`select *`

```
select *
from   Stuk;
```

Identieke operatie

Ook dit geeft een projectie, namelijk op alle kolommen. Dit `select`-statement zouden we in wiskundige termen de *identieke operatie* kunnen noemen: het laat de operand (brontabel) onveranderd.

Het componistnummer in de uitvoer is niet voor eindgebruikers bedoeld. In een eindgebruikersapplicatie blijft het verborgen en komt er de naam van de componist voor in de plaats. Om dat in SQL voor elkaar krijgen, moeten we een 'joinquery' uitvoeren op de tabellen Stuk en Componist. Met zo'n query kunnen we iedere Stuk-rij verbreden met de informatie uit de bijbehorende Componist-rij. In de volgende leereenheid gaan we daar uitgebreid op in.

2.2 CONSTANTE EN BEREKENDE RESULTAATKOLOMMEN

Constante in select-lijst

Een expressie in de `select`-lijst hoeft geen kolomnaam te zijn. Het mag ook een *constante* zijn, bijvoorbeeld een constant stukje tekst, een constant getal of een constante datum:

```
select componist, titel, 'heeft als niveau', niveau
from   Stuk;
```

Resultaat:

```
COMPONIST TITEL            CONSTANT         NIVEAU
========= ================ ================ ======
        1 Blue bird        heeft als niveau <null>
        2 Blue bird        heeft als niveau B
        4 Air pour charmer heeft als niveau B
        5 Lina             heeft als niveau B
        8 Berceuse         heeft als niveau <null>
        2 Cradle song      heeft als niveau B
        8 Non piu andrai   heeft als niveau <null>
        9 I'll never go    heeft als niveau A
       10 Swinging Lina    heeft als niveau B
        5 Little Lina      heeft als niveau A
       10 Blue sky         heeft als niveau A
```

Tekstconstante '…'

Een *tekstconstante* staat tussen enkele aanhalingstekens. Tekstconstanten mogen spaties en andere wittekens bevatten (tabs en einde-alineatekens). Deze zijn een letterlijk onderdeel van de tekstconstante; het zijn daarin tekens als alle andere.

Berekende expressie in select-lijst

De `select`-clausule mag ook expressies bevatten, waarvan de uitvoerwaarde wordt berekend uit kolomwaarden en/of constanten. In de berekening zelf mogen operatoren en/of functies voorkomen. Een eenvoudig voorbeeld:

```
select naam || ' is gevestigd in ' || plaats
from   Muziekschool;
```

Concatenatie

De 'berekening' is hier een *concatenatie*, zie voorbeeld 2.5 in leereenheid 2.

Resultaat:

```
CONCATENATION
=============================================
Muziekschool Amsterdam is gevestigd in Amsterdam
Reijnders' Muziekschool is gevestigd in Nijmegen
Het Muziekpakhuis is gevestigd in Amsterdam
```

Een resultaatkolom met berekende waarden krijgt van Firebird een 'technisch georiënteerde' kolomkop. Dit is geen probleem: dit soort uitvoer is niet voor eindgebruikers bedoeld. Toch kunnen we er wel wat aan doen, zoals we in de volgende paragraaf zullen zien.

2.3 ALIASKOLOMNAMEN

Kolomalias

Om in een resultaattabel een berekende kolom een minder technische naam te geven of een andere kolomnaam te wijzigen, kunnen we gebruikmaken van een *aliaskolomnaam*, kortweg *kolomalias*.

VOORBEELD 4.2

Geef een stukkenoverzicht, met stuknummer, titel en effectieve speelduur (die, naar wordt aangenomen, zo'n 10 procent hoger is dan de officiële speelduur).

```
select nr stuk,
       titel,
       speelduur * 1.1 effectieve_speelduur
from   Stuk;
```

Voor de duidelijkheid hebben we de kolomaliassen even vet gemaakt: kolom 'nr' heeft als alias 'stuk' en de berekende kolom heeft als alias 'effectieve_speelduur'.

`as`

Een alias wordt door een of meer spaties (of algemeen: door whitespace) van de expressie gescheiden. Ook kan het gereserveerde woord `as` worden gebruikt:

```
select nr as stuk,
       titel,
       speelduur * 1.1 as effectieve_speelduur
from   Stuk;
```

Resultaat:

```
    STUK TITEL           EFFECTIEVE_SPEELDUUR
======== ============== ====================
       1 Blue bird                      4.95
       2 Blue bird                      4.40
       3 Air pour charmer               4.95
       5 Lina                           5.50
       8 Berceuse                       4.40
       9 Cradle song                    3.85
      10 Non piu andrai                <null>
      12 I'll never go                  6.60
      13 Swinging Lina                  8.80
      14 Little Lina                    4.73
      15 Blue sky                       4.40
```

Een alias is een zogenaamde *identifier* en moet dus voldoen aan de syntactische beperkingen genoemd in de volgende paragraaf.

Kolomalias is een gewone kolomnaam

De resultaattabel van een `select`-query was tot nu toe een eindresultaat. In leereenheid 7 'Subselects en views' komen we ook `select`-expressies tegen die onderdeel uitmaken van een grotere, omvattende `select`-query. Het resultaat van de 'binnenste `select`' wordt dan gebruikt om mee verder te werken in de 'buitenste `select`'. Elke resultaattabel dient daarom conceptueel te worden opgevat als een tabel in de zin van het relationele model.

We zullen nog voorbeelden tegenkomen van dergelijke subselects waarbij een kolomalias van de 'binnenste `select`' door de `select`-query waarvan hij een onderdeel is, als een gewone kolomnaam wordt gebruikt.

2.4 GERESERVEERDE WOORDEN EN IDENTIFIERS

Een SQL-statement is op te vatten als een 'zin' met 'woorden' en symbolen. De 'woorden' zijn er in twee soorten:

Gereserveerd woord
Keyword

1 de woorden die tot de vaste taalschat behoren; deze worden *gereserveerde woorden* (Engels: *reserved words*, ook wel: *keywords*) genoemd
2 de woorden die door de programmeur zijn gekozen.

In het hiervoor gegeven `select`-statement zijn `select` en `from` gereserveerde woorden.

Identifier

Door de programmeur gekozen namen heten *identifiers*, en die moeten voldoen aan de volgende beperkingen:

1 Hun maximale lengte is 31 tekens.
2 Ze mogen alleen letters, cijfers en de underscore (_) bevatten.
3 Ze moeten beginnen met een letter.
4 Ze mogen niet overeenkomen met een gereserveerd woord.

Veel belangrijker dan deze 'harde' regels (die u vanzelf leert via foutmeldingen) zijn de volgende 'zachte' regels:
– Kies duidelijke namen, die de betekenis goed weerspiegelen.
– Wees consequent bij het kiezen van namen.
– Wees terughoudend in het kiezen van afkortingen; als u afkort, doe het dan volgens strakke conventies.

Afspraak gebruik hoofd- en kleine letters

Voor gelijksoortige objecten of eigenschappen hanteren we een vaste naamgeving- en spellingconventie. Zo schrijven we tabelnamen met één hoofdletter en kolomnamen in kleine letters.

2.5 OPMAAK

In deze cursus hanteren we vrij strikte opmaakconventies, gericht op de menselijke lezer. Zo zetten we een statement zoals

```
select *
from   Stuk;
```

vrijwel altijd op twee regels: met de `select`-clausule en de `from`-clausule elk op een aparte regel, en de eerste letter van de brontabel recht onder het sterretje (of het eerste teken van de eerste expressie in de `select`-clausule). In sommige gevallen is het juist weer duidelijker om ieder onderdeel van de `select`-clausule op een aparte regel te zetten, zoals in voorbeeld 4.2.

Voor de SQL-*interpreter* (het programmaonderdeel van het rdbms dat het statement verwerkt) maakt het niets uit; die is ook tevreden met de hele opdracht op één regel:

```
select * from Stuk;
```

of over vier regels verspreid:

```
select
*
from
Stuk;
```

Ook hoofd- en kleine letters maken voor de interpreter geen verschil. Voor de menselijke lezer wel, en daarom houden wij ons strikt aan onze eigen afspraken daarover.

Witteken

Spaties en 'returns' zijn, net als tabs, *wittekens* (Engels: *whitespaces*). Elk witteken is toegestaan als scheidingsteken tussen de 'woorden' en tekens van een statement. Als SQL-programmeur bewijst u uzelf en uw collega's echter een dienst door een strakke opmaak met vaste conventies. Automatiseringsbedrijven zullen hun werknemers daartoe dwingen, via een 'huisstijl'.

2.6 COMMENTAARCODE

Commentaar

Soms zullen we SQL-code verduidelijken of documenteren via *commentaar* in gewoon Nederlands. Commentaar is bedoeld voor de menselijke lezer en wordt niet verzonden naar het rdbms. Er zijn twee soorten commentaar: eenregelig of meerregelig.

`--`

Eenregelig commentaar begint met twee streepjes en wordt vaak gebruikt om een onderdeel van een SQL-statement te verduidelijken. Bijvoorbeeld:

```
select *         -- 2. Geef per rij alle kolomwaarden
from   Stuk;     -- 1. Ga uit van tabel Stuk: alle rijen
```

Conceptuele verwerkings-volgorde

In deze cursus zullen we commentaar soms 'misbruiken' voor didactische doeleinden, zoals in het voorgaande statement. Daar geven de nummers '1' en '2' in het commentaar aan wat de *conceptuele verwerkingsvolgorde* is, dat wil zeggen: in welke volgorde u de regels als mens kunt lezen om de juiste werking te begrijpen. Merk op dat u volgens het commentaar moet beginnen met de tweede regel: de `from`-clausule. Daar staat uit welke tabel de gegevens afkomstig zijn. De `select`-clausule zegt welke gegevens in het eindresultaat komen. In dit voorbeeld zijn dat alle kolomwaarden.

Meerregelig commentaar

`/* … */`

Script

Meerregelig commentaar staat tussen 'haakjes': `/*` en `*/`. Alle tekst daartussen is bedoeld voor de menselijke lezer. Het wordt meestal gebruikt om een SQL-*script* te documenteren. Zo'n script bevat een reeks SQL-opdrachten voor het uitvoeren van een grotere taak, bijvoorbeeld het creëren van alle tabellen van de Muziekdatabase. Omdat scripts worden bewaard, is het vaak zinvol er documentatie aan toe te voegen, zoals de auteur en de datum van de laatste wijziging.

2.7 DISTINCT

Wanneer we een resultaattabel zuiver relationeel opvatten, vormen de rijen een verzameling en zijn deze dus alle verschillend. In de praktijk blijkt SQL echter een halfslachtig soort tabellenalgebra, waarbij expressies lang niet altijd zuiver relationeel worden weergegeven of verwerkt. Neem als voorbeeld het volgende statement:

```
select genre, niveau
from   Stuk;
```

met als resultaat:

```
GENRE      NIVEAU
========== ======
jazz       <null>
jazz       B
klassiek   B
klassiek   B
klassiek   <null>
klassiek   B
klassiek   <null>
pop        A
jazz       B
klassiek   A
jazz       A
```

Hier staan 11 rijen. Interpreteren we de tabel echter als relatie, dan staat er een rijverzameling met maar 7 rijen. Voor weergave zonder meervoudige rijen, moeten we de `select`-lijst laten voorafgaan door een speciale operator, `distinct`:

`distinct`

```
select distinct genre, niveau
from   Stuk;
```

Resultaat:

```
GENRE      NIVEAU
========== ======
jazz       <null>
jazz       A
jazz       B
klassiek   <null>
klassiek   A
klassiek   B
pop        A
```

Welbeschouwd is dit geen overzicht van stukken, maar van genre-niveau-combinaties *als zelfstandige nieuwe objecten*. Dat overzicht zouden we (conceptueel gezien) kunnen uitbreiden met een kolom voor het aantal stukken bij zo'n combinatie. In leereenheid 6 'Statistische informatie' zullen we dat ook werkelijk doen, zij het via een heel ander mechanisme.

Let op!

De toevoeging `distinct` heeft altijd betrekking op de volledige `select`-lijst. De volgende `select`-query is dan ook zonder betekenis en geeft een foutmelding:

```
select genre, distinct niveau   -- fout!
from   Stuk;
```

Over het algemeen bevat een `select`-lijst een kandidaatsleutel en zijn de rijen dus bij voorbaat uniek. Het gebruik van `distinct` maakt dan voor het resultaat niets uit en dient vermeden te worden.

Rijen en tupels
In Inleiding informatica hebben we al gezien dat een tabel overeenkomt met het wiskundige begrip 'relatie', en dat daar de naam 'relationele database' vandaan komt. Zoals de term *rij* zich verhoudt tot *tabel*, zo verhoudt de term *tupel* zich tot *relatie*: tupels zijn de 'rijen' van een relatie, in een meer formeel/wiskundige betekenis. De tupels van een relatie vormen een verzameling; ze zijn dus alle verschillend en hebben geen volgorde. Een veelgebruikte naam voor de 'tupelpopulatie' is *extensie*.
Wanneer we de termen rij en tabel zuiver relationeel interpreteren, is er geen verschil met 'tupel' en 'relatie'. Helaas maakt de aard van de taal SQL het wenselijk om toch onderscheid te maken. Al zullen we steeds ons best doen om SQL als een relationele taal te zien en SQL-resultaattabellen als relaties te interpreteren, dit zal niet altijd houdbaar zijn. SQL blijkt maar ten dele relationeel; we zullen meermalen voorbeelden zien waarbij volgorde wel degelijk een rol speelt en waarbij gelijke rijen voorkomen die los van elkaar worden behandeld.

Distinct: een pseudo-operator
Wanneer SQL een consequent relationele taal was, zou de 'operator' `distinct` niet nodig zijn. Vanuit relationeel standpunt (waarbij we abstraheren van gelijke rijen in de tabelweergave) doet `distinct` immers helemaal niets! We noemen `distinct` dan ook bij voorkeur een *pseudo-operator*, die wordt gebruikt om niet-relationele trekjes van SQL te corrigeren, wanneer dat zo uitkomt. In het vervolg van deze cursus zullen we er nog af en toe op wijzen dat het gebruik van `distinct` soms wel praktisch is (en mogelijk ook tot relatief snelle verwerking leidt), maar conceptueel gezien niet mooi. Het kan ook altijd zonder.

OPGAVE 4.2
Geef een `select`-statement voor alle werkelijk in stukken gebruikte muziekinstrumenten (instrumentnaam met toonhoogte), elk instrument één keer.

3 Datatypen

Datatype

De gegevens in een kolom zijn van één bepaald *datatype*, dat in de tabeldefinitie (`create table`) moet worden opgegeven. Het onderwerp datatype komt daarom terug in leereenheid 9 over gegevensstructuren. Vaak echter moeten we ook bij het gebruik van gegevens, via SQL, rekening houden met het datatype van een kolom. Om een voorbeeld te geven: wanneer kalenderdata als `date` zijn opgeslagen, kunnen we er 'datumrekenkunde' mee bedrijven. Intern worden ze dan in een of ander numeriek formaat opgeslagen. Zo kun je bijvoorbeeld twee kalenderdata van elkaar aftrekken om het verschil in dagen te vinden. Slaan we ze als tekst op, dan zijn zulke berekeningen onmogelijk.

3.1 DATATYPEN VAN DE SQL-STANDAARD

Datatypen van SQL2

Figuur 4.3 geeft een overzicht van enkele belangrijke datatypen van de SQL-standaard. Deze worden door de meeste rdbms'en ondersteund.

datatype	*voor welke gegevens?*	*toelichting*
`integer`	gehele getallen	–2.147.483.648 t/m 2.147.483.647
`numeric(`*n*`, `*m*`)` of `numeric(`*n*`)`	decimale getallen met vast aantal decimalen (*fixed point*)	*n* = lengte inclusief decimalen (1 t/m 15) *m* = aantal decimalen voorbeelden: - voor 123.45 is *n* = 5 en *m* = 2 - voor -1234 is *n* = 4 en *m* = 0 voor gehele getallen volstaat `numeric(`*n*`)`, dus: `numeric(4,0)` = `numeric(4)`.
`char(`*n*`)`	tekst (vaste lengte)	*n* = aantal tekens
`varchar(`*n*`)`	tekst (variabele lengte)	*n* = maximum aantal tekens

`date`
`time`
`timestamp`
	datum
	tijd
	datum-met-tijd
		opgeslagen in intern numeriek formaat

`blob`	binaire objecten	plaatjes, geluid etc.

FIGUUR 4.3 Enkele belangrijke SQL-datatypen

Tekstwaarden
De twee teksttypen zijn voor kortere of langere stukjes tekst. Voor een `char`-waarde wordt een vaste hoeveelheid geheugen gereserveerd; niet-gebruikte ruimte wordt opgevuld met spaties. Voor een `varchar`-waarde wordt geheugen op maat gereserveerd.

Alfanumerieke waarde
Tekenrij
String
Character

Een tekstwaarde heet ook *alfanumerieke waarde*, *tekenrij* of *string*. We gebruiken deze termen door elkaar. Eén enkel teken heet een *character*. Zie verder paragraaf 3.2.

Datum- en tijdwaarden
Gebruikers (waaronder ook de programmeur van een SQL-script) zijn gewend datum- en datum/tijdwaarden te schrijven en lezen als strings, terwijl het rdbms er speciale datatypen voor heeft: `date` (voor kalender-data), `time` (voor de tijden van een 24-uursklok) en `timestamp` (voor datum/tijdwaarden, een combinatie van `date` en `time`). Daarmee is het volgende probleem gesteld: hoe houdt het rdbms en hoe houden programmeurs `date`-, `time`- en `timestamp`-waarden enerzijds en stringwaarden anderzijds uit elkaar?

Bij de installatie van het rdbms wordt meestal een standaard-weergaveformaat ingesteld. Ook de taal (voor onder andere de namen van dagen en maanden) wordt daarbij vastgelegd. Het standaardformaat voor datumweergave door Firebird is yyyy-mm-dd (let op: dit is de Engelse volgorde, dus eerst de maand en dan de dag).

yyyy-mm-dd

Als we met de opdracht

```
select current_date
from   Rdb$database;
```

de 'huidige datum' opvragen op 18 mei 2016 krijgen we als resultaat

```
CURRENT_DATE
============
2016-05-18
```

`current_date`

Hierin is `current_date` een ingebouwde SQL-functie, onderdeel van de SQL-standaard.

Datuminvoer

Ditzelfde formaat gebruiken we ook in SQL-code op plaatsen waar een datumwaarde wordt verwacht: we schrijven dan letterlijk de string '2016-05-18'. Een voorbeeld:

```
insert into Componist values (23, 'Johan de Meij', '1953-11-23', null);
```

Firebird kan ook allerlei andere weergaven lezen, zoals 'May 18, 2016' en '18-may-2016', maar als *uitvoer* geeft Firebird standaard '2016-05-18'. Daarom gebruiken wij dat formaat ook zoveel mogelijk voor *invoer* van datumwaarden.

Overigens is het voor een Nederlandse applicatie natuurlijk wenselijk dat uitvoer en invoer als '18 mei 2016' (ook) mogelijk zijn. Dat kan geregeld worden, maar wij doen dat in deze cursus niet. In paragraaf 5 vertellen we er iets meer over.

Lay-out
De manier waarop gegevens worden afgedrukt, is afhankelijk van het datatype:
– numerieke gegevens worden rechts uitgelijnd
– alfanumerieke gegevens worden links uitgelijnd
– kalenderdata worden links uitgelijnd.

3.2 CHARACTER SETS

Character set

Bij de alfanumerieke datatypen kan een *character set* worden gespecificeerd: een gestandaardiseerd alfabet van zichtbare (dat zijn de meeste) en onzichtbare tekens. De character set kan worden ingesteld voor een hele database, voor een tabel of zelfs voor een kolom. Er bestaan vele character sets, bijvoorbeeld ook voor Chinese characters. Per character set bestaan vaak nog weer verschillende sorteervolgorden (*collation orders*).

Collation order

Een veelgebruikte character set is ISO-8859-1, zie figuur 4.4. Het is een 8-bits code, wat leidt tot $2^8 = 256$ verschillende tekens of codes. Ze zijn genummerd van 0 t/m 255.

ASCII

De deelset van de characters 0 t/m 127 van ISO-8859-1 (en van vele andere character sets) is identiek aan de welbekende *ASCII*-set (ASCII = American Standard Code for Information Interchange). Deze bevat niet alleen afdrukbare tekens, maar ook niet-afdrukbare tekens en stuurcodes.

In de tabel zijn deze door een lettercode weergegeven, bijvoorbeeld: BEL (het belsignaal), NL ('newline', overgang op een nieuwe regel), CR ('carriage return', terugkeer naar begin van de regel), ESC ('escape') en DEL ('delete'). Vermeldenswaardig is ook SP (= ASCII 32, de spatie), dit is een character als alle andere, al is het 'onzichtbaar'.

Wanneer voor de database géén character set is gedefinieerd, worden bij de communicatie van client richting server de character codes (in 'nullen en enen') ongewijzigd opgeslagen. Dit kan problemen geven wanneer iemand een afwijkende client gebruikt (niet ondenkbaar bij internet-applicaties). Op de server is de code voor een 'é' dan bijvoorbeeld 130, terwijl hij op die afwijkende client 145 is, en 130 daar een heel ander character voorstelt.

Language driver

Als er wel een character set is gespecificeerd, wordt het character via een *language driver* 'gemapt' op het juiste character uit de character set van de database en vice versa. Zo wordt het character op alle clients goed afgebeeld.

In geen van de voorbeelddatabases in de cursus is een character set gespecificeerd. In plaats daarvan hebben we gewoon geen accenten in de tekstvelden gebruikt: stuk 10 heet eigenlijk 'Non più andrai', maar we schrijven 'Non piu andrai'.

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	NL	11	VT	12	NP	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	DEL
128	--	129	--	130	--	131	--	132	--	133	--	134	--	135	--
136	--	137	--	138	--	139	--	140	--	141	--	142	--	143	--
144	--	145	--	146	--	147	--	148	--	149	--	150	--	151	--
152	--	153	--	154	--	155	--	156	--	157	--	158	--	159	--
160	¨	161	¡	162	¢	163	£	164	¤	165	¥	166	¦	167	§
168	°	169	©	170	ª	171	«	172		173	µ	174	®	175	¯
176		177	±	178	²	179	³	180	´	181	½	182		183	·
184	¸	185	¹	186	º	187	»	188	¼	189	Â	190	¾	191	¿
192	È	193	Á	194	Â	195	Ã	196	Ä	197	Í	198	Æ	199	Ç
200	Ð	201	É	202	Ê	203	Ë	204	Ì	205	Õ	206	Î	207	Ï
208	Ø	209	Ñ	210	Ò	211	Ó	212	Ô	213	Ý	214	Ö	215	×
216	à	217	Ù	218	Ú	219	Û	220	Ü	221	å	222	Þ	223	ß
224	è	225	á	226	â	227	ã	228	ä	229	í	230	æ	231	ç
232	ð	233	é	234	ê	235	ë	236	ì	237	õ	238	î	239	ï
240	ø	241	ñ	242	ò	243	ó	244	ô	245	ý	246	ö	247	÷
248		249	ù	250	ú	251	û	252	ü	253		254	þ	255	ÿ

FIGUUR 4.4 De character set ISO-8859-1, met ASCII als deelset

3.3 TYPECASTING

Converteren

Typecasting

Soms is het nodig een waarde van een bepaald datatype te *converteren* naar een corresponderende waarde van een ander datatype, bijvoorbeeld van `numeric` naar `varchar`, of van `date` naar `varchar`. We noemen dit: *typecasting*, of kortweg: *casting*. Firebird heeft hiervoor de `cast`-functie, die in paragraaf 5.3 wordt behandeld.

4 Operatoren

Operator en operand

Binaire en unaire operator

Een *operator* is een teken of een samenstel van tekens waardoor een bewerking wordt gedefinieerd. Een operator werkt op een of meer *operanden* en 'geeft een resultaatwaarde terug'. Zo kan het minteken worden gebruikt om twee getallen (de operanden) van elkaar af te trekken, bijvoorbeeld: 5 – 3. Dit is een *binaire operator*, omdat hij twee operanden heeft. Het minteken kan ook een *unaire operator* voorstellen, werkend op één operand: de unaire min-operator. Deze neemt van een getal het tegengestelde, bijvoorbeeld: –3. Zie tabel 4.1 voor een overzicht van operatoren, die onder meer in expressies van de `select`-lijst mogen worden gebruikt.

In een `where`-clausule mogen nog andere typen operatoren worden gebruikt, waaronder vergelijkingsoperatoren (zoals > en <) en logische operatoren (`and`, `or` en `not`).

TABEL 4.1 Operatoren

datatype	*operator*	*betekenis*
numeriek	+ (binair)	optelling
	+ (unair)	getal zelf
	– (binair)	aftrekking
	– (unair)	tegengestelde
	*	vermenigvuldiging
	/	deling
alfanumeriek	||	stringconcatenatie
datum/tijd	–	verschil van datum/tijd-waarden
datum/tijd met numeriek	+	getal (aantal dagen) opgeteld bij datum/tijd
	–	getal (aantal dagen) afgetrokken van datum/tijd

In de volgende paragrafen gaan we wat dieper in op het werken met deze operatoren.

4.1 NUMERIEKE OPERATOREN

Numerieke operatoren

+, -, *, /

Numerieke expressies mogen worden gevormd door de numerieke operatoren optellen, aftrekken, vermenigvuldigen en delen: de operatoren +, – , * en / . Hierbij is / de gehele deling: 7 / 3 geeft 2, ofwel het geheel aantal keer dat 3 in 7 past.

VOORBEELD 4.3

```
select 6 + 28 / 7 * 3 - 1 berekening
from   Rdb$database;
```

Resultaat:

```
   BEREKENING
=============
           17
```

Toelichting
Vermenigvuldigen en delen vinden plaats in de volgorde waarin ze voorkomen en hebben voorrang op optellen en aftrekken. Optellen en aftrekken vinden eveneens plaats in de volgorde waarin ze voorkomen. Eerst wordt dus 28 / 7 * 3 berekend (geeft 12) en vervolgens 6 + 12 – 1 (geeft 17). Dit komt overeen met de normale voorrangsregels (*prioriteit* van de operatoren). Om een andere volgorde af te dwingen, moet u haakjes gebruiken.

Prioriteit

De optelling en de aftrekking komen ook voor als unaire operator, bijvoorbeeld in +7 + (–3). Hierin zijn de eerste plus en het minteken unair.

4.2 ALFANUMERIEKE OPERATOREN

Alfanumerieke operator

Een *alfanumerieke operator* werkt op alfanumerieke operanden. Bijvoorbeeld: 'vliegtuig' of 'ak34c2'. Andere namen voor een alfanumerieke waarde zijn *string* en *tekenrij*. Er is maar één alfanumerieke operator: de *concatenatie* (||).

Concatenatie ||

Deze rijgt strings aaneen (concatenatie = 'samenketening'). Hiervan hebben we al een paar voorbeelden gezien.

Stringoperator

Een alfanumerieke operator wordt ook *stringoperator* genoemd.

4.3 DATUM- EN TIJDOPERATOREN

Datumoperatoren

Datum- en tijdwaarden worden opgeslagen in een intern numeriek formaat, waardoor met deze waarden gerekend kan worden. Firebird heeft één zuivere datumoperator: datumwaarden van elkaar aftrekken. De uitkomst is het aantal dagen verschil.

VOORBEELD 4.4 (aftrekking kalenderdata)

Geef van elke componist de naam en de leeftijd – op de systeemdatum – in dagen.

Oplossing:

```
select naam   componist,
       current_date - geboortedatum   leeftijd_in_dagen
from   Componist;
```

Resultaat (bij systeemdatum 13 mei 2016):

```
COMPONIST           LEEFTIJD_IN_DAGEN
=================== ================
Charlie Parker                 40695
Thomas Guidi                   18391
Rudolf Escher                  38112
Sofie Bergeijk                 13089
W.A. Mozart                    95070
Karl Schumann                  22131
Jan van Maanen                 11205
```

Overigens valt bij de term 'leeftijd' bij sommige componisten wel een vraagteken te zetten ...

De eerste dag
Elk SQL-dialect heeft zijn eigen 'eerste dag'. Gebruikers hoeven die over het algemeen niet te weten. Toch is het misschien aardig er een paar te vermelden: de wereld van Firebird begint op 1 januari van het jaar 100, die van Oracle begint op 1 januari van het jaar –4753 (dus voor Chr.). Er zijn ook 'laatste dagen': Firebird kan vooruit tot 29 februari 32768, Oracle tot 31 december 4712.

4.4 GEMENGDE OPERATOREN

Bij een datum kan een getal worden opgeteld of afgetrokken. Dat getal is een tijdsperiode, uitgedrukt in dagen. De uitkomst is weer een datum:

```
select current_date,     -- vandaag
       current_date + 7, -- volgende week
       current_date - 7  -- vorige week
from   Rdb$database;
```

Hieronder het resultaat van dit commando, toen het werd gegeven op 13 mei 2016:

```
CURRENT_DATE         ADD    SUBTRACT
============ =========== ===========
2016-05-13   2016-05-20  2016-05-06
```

Gemengde operatoren

Aangezien bij deze optelling en aftrekking de operanden van verschillend datatype zijn, spreken we van *gemengde operatoren*.

OPGAVE 4.3
Geef alle muziekinstrumenten met de toonhoogte en de instrumentnaam aaneengeschreven, bijvoorbeeld: 'sopraansaxofoon', 'altfluit', 'piano'.

OPGAVE 4.4
Welke datum is het over 1000 dagen?

OPGAVE 4.5
Geef van alle stukken: componistnummer, titel en niveaucode. Wanneer een stuk geen niveaucode heeft, moet in plaats daarvan de tekst 'geen speelstuk' worden afgedrukt.

5 Functies

Interne functie

User defined function

In de wiskunde kennen we naast operatoren ook *functies*. Ook SQL-expressies kunnen functies bevatten. Elk SQL-dialect kent *interne functies*, als onderdeel van de taaldefinitie. Bovendien bieden moderne dialecten de mogelijkheid aan de programmeur om zelf functies toe te voegen, zogenaamde udf's (*user defined functions*). In deze paragraaf geven we een overzicht van de belangrijkste interne functies en voorbeelden van de toepassing ervan.

5.1 WAT IS EEN FUNCTIE?

Functie, resultaat-waarde en argumenten

Een *functie* heeft veel gemeen met een operator. Net als een operator is het een voorschrift voor het uitvoeren van een berekening, die een resultaatwaarde oplevert op de plek waar zij wordt *aangeroepen*.

Een verschil met een operator is dat een functie een eigen naam heeft, die wordt gevolgd door de waarden waarmee gerekend moet gaan worden: de *argumenten*. De argumenten staan tussen haakjes, meestal als kommalijst, maar soms met een afwijkende syntaxis. Er kunnen nul, een of meer argumenten zijn. Daarbij hoeft u niet alleen te denken aan 'echt' rekenen (zoals bij allerlei wiskundige functies: kwadrateren, worteltrekken, ...) maar ook aan bewerkingen met tekenrijen of kalenderdata.

round-functie

VOORBEELD 4.5

De numerieke afrondingsfunctie `round`. Voorbeelden van een aanroep van `round`:

`round(76.543)`	resultaat: 77	(afronden op gehele waarde)
`round(76.543, 1)`	resultaat: 76.500	(afronden op 'één decimaal')

We zien dat de functie `round` één of twee argumenten heeft. Het eerste is de getalwaarde. Het optionele tweede argument geeft aan op welk aantal decimalen moet worden afgerond. Volgens de Firebird-handleiding heeft, als er een tweede argument aanwezig is, het resultaat 'gewoonlijk' evenveel decimalen als het eerste argument. Vandaar dat in het voorbeeld `round(76.543, 1)` niet 76.5 geeft maar 76.500.

Geretourneerde waarde

De resultaatwaarde van een functie heet de *geretourneerde waarde* (Engels: *return value*). Die waarde wordt als het ware 'teruggegeven' door de functie-expressie. Met de geretourneerde waarde kan worden doorgerekend in een grotere expressie. Bijvoorbeeld:

`5 + round(876.543)` resultaatwaarde: 882

Functies kunnen worden gebruikt in diverse clausules van SQL-statements. Bijlage 1 achter in cursusdeel 2 geeft een overzicht, met voorbeelden.

5.2 DATUM- EN TIJDFUNCTIES

We hebben het al gehad over de opslag en weergave van datum- en tijdwaarden, van de datatypen `date`, `time` en `timestamp`. Firebird converteert automatisch iedere string in het juiste format naar de bijbehorende `date`-, `time`- of `timestamp`-waarde, als die string tenminste op een plek in de code staat waar (respectievelijk) *alleen maar* een `date`-, `time`- of `timestamp`-waarde mag staan. Andersom drukt Firebird automatisch iedere `date`-, `time`- of `timestamp`-waarde af als string.

Zulke conversies kunnen ook expliciet gedaan worden door een udf. De SQL-programmeur kan dan zelf bepalen welk format hij wil gebruiken, bijvoorbeeld met Nederlandse maandnamen. Wij gebruiken in deze cursus geen udf's.

`extract`

Er is wel een interne Firebird-functie die soms van pas komt bij het werken met datumwaarden: `extract`. De opdracht

```
select naam, extract(year from geboortedatum) geboortejaar
from   Componist;
```

geeft bijvoorbeeld als resultaat

```
NAAM                 GEBOORTEJAAR
==================== ============
Charlie Parker               1904
Thomas Guidi                 1966
Rudolf Escher                1912
Sofie Bergeijk               1980
W.A. Mozart                  1756
Karl Schumann                1955
Jan van Maanen               1985
```

In plaats van `year` kunnen we onder andere ook `day`, `weekday`, `month` en `minute` opvragen. Raadpleeg de Firebird-documentatie voor andere mogelijkheden.

Verder hebben we `current_date` al genoemd; dit is een datumfunctie zonder argumenten, die (dus) zonder haakjes geschreven wordt.

5.3 DE CAST-FUNCTIE

Typecastfunctie

`cast`

Firebird beschikt over een speciale typecastfunctie `cast`, voor het omzetten van een waarde van het ene datatype naar een waarde van een ander datatype.

VOORBEELD 4.6

Geef van alle muziekstukken het stuknummer en de speelduur in één doorlopende zin: 'stuk nr. ... duurt ... minuten.'

Hier wordt gevraagd strings en numerieke waarden te concateneren. Aangezien concatenatie een stringoperatie is, is conversie nodig van de numerieke waarden naar een stringtype:

```
select 'stuk ' || cast(nr as varchar(4)) || ' duurt ' ||
       cast(speelduur as varchar(5)) || ' minuten.'   speelduur
from   Stuk;
```

Merk op dat de aanroep van de `cast`-functie een speciale syntaxis heeft, met '`as`' op de plaats waar een normale functieaanroep een komma heeft. Merk verder op dat het eerste voorkomen van 'speelduur' een kolomnaam is, het tweede een alias.

Resultaat:

```
SPEELDUUR
============================
stuk 1 duurt 4.5 minuten.
stuk 2 duurt 4.0 minuten.
stuk 3 duurt 4.5 minuten.
stuk 5 duurt 5.0 minuten.
stuk 8 duurt 4.0 minuten.
stuk 9 duurt 3.5 minuten.
<null>
stuk 12 duurt 6.0 minuten.
stuk 13 duurt 8.0 minuten.
stuk 14 duurt 4.3 minuten.
stuk 15 duurt 4.0 minuten.
```

Zie leereenheid 2, voorbeeld 2.5.

Stuk 10 heeft geen speelduur. Aangezien een concatenatie met een null ook als resultaat een null geeft, verschijnt daarvoor <null> in de uitvoer.

Dit was de 'nette' oplossing. Zonder typecasting blijkt het echter ook goed te gaan:

```
select 'stuk ' || nr || ' duurt ' || speelduur || ' minuten.'  speelduur
from   Stuk;
```

Impliciete typecasting

Firebird blijkt bij concatenatie de typecasting dus *impliciet*, ofwel automatisch uit te voeren. Wel zo gemakkelijk!

Het converteren van een string die een datum representeert (in een van de geaccepteerde formaten) naar een datumwaarde gebeurt alleen automatisch als die string op een plek in de code staat waar alleen maar een datumwaarde mag staan. In de volgende opgave willen we rekenen met letterlijke datumwaarden. Dat zijn echter strings, en daar mogen we niet zomaar mee rekenen: we moeten ze expliciet casten naar `date`'s.

OPGAVE 4.6

In het jaar 1582 werd door paus Gregorius XIII de schrikkeljaartelling volgens de Juliaanse kalender (jaartal deelbaar door 4, dan schrikkeljaar) vervangen door de Gregoriaanse kalender (jaartal deelbaar door 4, dan schrikkeljaar, behalve als het een 100-voud is dat geen 400-voud is). Ter correctie van de 'uitloop' gedurende vele eeuwen werden de data 5 tot en met 14 oktober 1582 overgeslagen. Op 4 oktober 1582 volgt dus 15 oktober 1582. Ga na of de SQL-datumrekenkunde hiermee rekening houdt.

Cast van null

Cast van null

Soms zullen we `cast` loslaten op `null`, om de van zichzelf typeloze `null` naar een bepaald datatype te converteren. Bijvoorbeeld: `cast(null as date)` maakt van `null` een `date`-null, en `cast(null as char(20))` maakt van `null` een `char(20)`-null. Dat lijkt wat vreemd, maar op sommige plaatsen wordt nu eenmaal een waarde van een specifiek datatype verwacht. Staat op die plaats `null`, dan is conversie nodig.

5.4 DE FUNCTIES CASE EN IIF

Twee bijzondere functies, waaraan we later in de cursus nog plezier zullen beleven, zijn de functies `case` en `iif`. Ze komen van pas in situaties waarbij een keuze moet worden gemaakt uit twee of meer alternatieven.

Het volgende voorbeeld illustreert de functie `case`.

VOORBEELD 4.7

Geef een overzicht van stukken met nr, titel en de waarde 'niet ingevuld', 'kort', 'middel' of 'lang'. Hierbij verklaren we stukken met speelduur <= 5 tot 'kort', die met speelduur > 5 en < 10 tot 'middel' en de overige stukken met ingevulde speelduur tot 'lang'.

Oplossing:

```
select nr, titel, case
                    when speelduur is null then 'niet ingevuld'
                    when speelduur <= 5 then 'kort'
                    when speelduur > 5 and speelduur < 10 then 'middel'
                    else 'lang'
                  end
from Stuk;
```

Toelichting
De case-expressie bevat drie when-clausules. Elke when-clausule bevat een logische conditie. De eerste daarvan die *true* is bepaalt welke waarde (achter then) wordt geretourneerd. Is geen ervan *true*, dan wordt de waarde na else geretourneerd.

Resultaat:

```
      NR TITEL          CASE
======== ============== ======
       1 Blue bird      kort
       2 Blue bird      kort
      ..
      13 Swinging Lina  middel
      14 Little Lina    kort
      15 Blue sky       kort
```

Algemene vorm van case-aanroep
De algemene vorm van een case-aanroep is als volgt:

```
case [testexpressie]
  when expressie then resultaat
  [when expressie then resultaat …]
  [else defaultresultaat]
end
```

Toelichting
– Wat tussen haken ([ ]) staat, is optioneel. Er zijn dus één of meer when-clausules, een optionele testexpressie en een optionele else-clausule.
– De testexpressie, indien aanwezig, wordt op het rijtje af met de when-expressies vergeleken. Bij een match (gelijkheid) retourneert de case-functie de betreffende resultaatwaarde. Is er geen match en is er een else-clausule dan wordt het defaultresultaat geretourneerd. Is er geen match en ook geen else-clausule, dan wordt null geretourneerd.
– Indien de testexpressie ontbreekt, mag *true* gelezen worden. In dat geval moeten de when-expressies logische condities zijn.

De functie iif
De functie iif dient om een keuze te maken uit twee alternatieven. Deze functie wordt als volgt aangeroepen:

```
iif(conditie, resultaat1, resultaat2)
```

Indien *conditie* geldt wordt *resultaat1* geretourneerd en anders *resultaat2*.
Een iif-aanroep is een bijzonder geval van een case-aanroep.
Bovenstaande aanroep is equivalent met de volgende aanroep van case:

```
case
  when conditie then resultaat1
  else resultaat2
end
```

VOORBEELD 4.8

Geef een overzicht van stukken met stuknummer, titel en de waarde 'kort' of 'lang'. Hierbij verklaren we stukken met speelduur <= 5 tot 'kort' en alle overige stukken tot 'lang'.

Oplossing:

```
select nr, titel, iif(speelduur <= 5, 'kort', 'lang')
from   Stuk;
```

Resultaat:

```
      NR TITEL           CASE
======== ============== ======
       1 Blue bird      kort
       2 Blue bird      kort
      ..
      13 Swinging Lina  lang
      14 Little Lina    kort
      15 Blue sky       kort
```

Let op de kop van de derde resultaatkolom. Hieruit blijkt dat een `iif`-aanroep door Firebird daadwerkelijk wordt vertaald naar een aanroep van `case`.

5.5 USER DEFINED FUNCTIONS

User defined function
Udf

SQL-dialecten wijken onderling nogal af op het gebied van interne functies. Door de mogelijkheid zelf functies te definiëren als *user defined function* (*udf*) zijn deze verschillen minder belangrijk. De eigenlijke functiedefinitie vindt plaats met behulp van een gastheertaal (*host language*) zoals C of C++. Indien u wilt weten hoe een en ander voor Firebird precies in zijn werk gaat, raadpleeg dan de Developer's guide.

6 Selecties: where

In paragraaf 2.1 werd de *projectie* geïntroduceerd: een vorm van de `select`-operator waarbij uit de brontabel een bepaalde deelverzameling van kolommen wordt genomen. Ook zagen we dat in de `select`-clausule constante of berekende resultaatkolommen mogen worden opgenomen. In het standaardgeval bevat de projectie-kolomverzameling een kandidaatsleutel, waardoor elke rij van de brontabel één rij in de `select`-resultaattabel oplevert. Zie in dit verband de bespreking van de pseudo-operator `distinct` in paragraaf 2.

We zullen nu voorwaarden opleggen aan de rijen, in een extra clausule met `where`. Alleen rijen die aan de voorwaarden voldoen, dragen bij aan het `select`-resultaat.

6.1 VOORWAARDEN AAN RIJEN

Selectie

Wanneer we de resultaattabel beperken tot een verzameling rijen die aan een bepaalde voorwaarde voldoet, spreken we van een *selectie*. Hiertoe dient de `where`-clausule, met een *selectieconditie*.

VOORBEELD 4.9
(gecombineerde selectie en projectie)

```
select componist, titel, niveau        -- 3: projectie
from   Stuk                            -- 1: brontabel
where  genre = 'klassiek';             -- 2: selectie
```

Zie figuur 4.5. Let op de volgorde, gegeven door het genummerde commentaar: uitgaande van de brontabel vindt eerst selectie plaats (dit geeft de rijen met het juiste genre), daarna projectie (dit beperkt de tussenresultaattabel tot de kolommen uit de `select`-lijst.

Stuk

nr	componist	titel	origineel	genre	niveau	speelduur	jaar
1	1	Blue bird		jazz		4.5	1954
2	2	Blue bird	1	jazz	B	4.0	1988
3	4	Air pour charmer		klassiek	B	4.5	1953
5	5	Lina		klassiek	B	5.0	1979
8	8	Berceuse		klassiek		4.0	1786
9	2	Cradle song	8	klassiek	B	3.5	1990
10	8	Non piu andrai		klassiek			1791
12	9	I'll never go	10	pop	A	6.0	1996
13	10	Swinging Lina	5	jazz	B	8.0	1997
14	5	Little Lina	5	klassiek	A	4.3	1998
15	10	Blue sky	1	jazz	A	4.0	1998

```
select componist, titel, niveau
from   Stuk
where  genre = 'klassiek'
```

componist	titel	niveau
4	Air pour charmer	B
5	Lina	B
8	Berceuse	
2	Cradle song	B
8	Non piu andrai	
5	Little Lina	A

FIGUUR 4.5 Selectie gevolgd door projectie

Het kan verwarrend zijn dat de selectievoorwaarde is geassocieerd met `where` en de projectie met `select`. Dit heeft een historische reden.

6.2 VERGELIJKINGSOPERATOREN

Vergelijkings-operator

. We gebruiken de volgende *vergelijkingsoperatoren*.

=	is gelijk aan
>	is groter dan
>=	is groter dan of gelijk aan
<	is kleiner dan
<=	is kleiner dan of gelijk aan
<>	is ongelijk aan

Al deze operatoren kunnen numerieke operanden, stringoperanden én datum-/tijdoperanden hebben.

Groter-gelijk zichzelf?
De operatoren >= en <= zorgen soms toch voor verwarring, met name wanneer constante waarden met elkaar worden vergeleken. Neem bijvoorbeeld de volgende logische expressie (bewering):

3 >= 3

Dit is een ware bewering, zoals blijkt uit de volgende herschrijving met een logische `or`:

3 >= 3 ⇔ (3 > 3) `or` (3 = 3) eerste operand van `or` is *false*, tweede is *true*

Aangezien `or` en/of betekent, is het voldoende dat één van beide operanden *true* is, om de bewering als geheel *true* te maken. Evenzo is 4 >= 3 een logische expressie met waarde *true*.

ASCII-ordening voor strings

Ordening
De ordening van numerieke waarden is de gebruikelijke. Bij string-operanden wordt gebruikgemaakt van de ingestelde ordening (*collation order*) van de character set, zie paragraaf 3.2. Voor de eerste 128 karakters van de character set is dat altijd overeenkomstig ASCII. De volgende expressies bijvoorbeeld leveren allemaal ware beweringen op:

```
'schip' < 'schipbreuk'
'schip' > 'boot'
'schip' <> 'boot'
'Schip' < 'schip'
```

Bij datum/tijdoperanden vindt de vergelijking plaats op basis van 'vroeger of later'. Bijvoorbeeld:

```
cast('31-dec-1999' as date) < cast('01-jan-2000' as date)
```

levert een ware bewering op. De expliciete casts van string naar datumtype zijn hier nodig, want anders zouden de datumstrings als zuivere tekenrijen vergeleken worden. Bij dit voorbeeld levert dat iets heel anders op:

```
'31-dec-1999' < '01-jan-2000'
```

is een onware bewering. Immers: de '3' heeft een hogere ASCII-waarde dan de '0', zodat '31-dec-1999' in het 'ASCII-woordenboek' op een hogere plaats terecht komt dan '01-jan-2000'.

OPGAVE 4.7
Probeer de twee datumvoorbeelden uit. Gebruik `iif` om te laten zien wat de waarde van een vergelijking is.

6.3 DE OPERATOR BETWEEN ... AND

between ... and

Met de operator 'between ... and' kunnen we een selectie uitvoeren op basis van waarden in een interval, inclusief de grenzen. Een voorbeeld:

```
select naam, geboortedatum
from   Componist
where  geboortedatum between '1700-01-01' and '1799-12-31';
```

De voorwaarde is gelijkwaardig met: `geboortedatum >= '1700-01-01' and geboortedatum <= '1799-12-31'`. Het resulaat bestaat uit alle componisten die geboren zijn in de achttiende eeuw:

```
NAAM                GEBOORTEDATUM
=================== =============
W.A. Mozart         1756-01-27
```

Ternaire operator

Merk op dat … between … and … een operator is met drie invulplekken: een *ternaire operator*. De operanden mogen van elk datatype zijn waarvoor een ordening is gedefinieerd: numeriek, alfanumeriek of datum.
De syntaxis van between ... and illustreert hoezeer men gepoogd heeft SQL op gewoon Engels te laten lijken. De beperking tot intervallen inclusief de grenswaarden heeft daar alles mee te maken.

De ontkenning van een between-conditie is op twee manieren mogelijk:
– met een expliciete not-operator: not(... between ... and ...)
– met de ternaire not between ... and-operator.

6.4 DE OPERATOR LIKE

Operator like voor zoeken op deel-string

Wanneer we rijen willen zoeken en selecteren op delen van strings in één kolom, biedt de operator like uitkomst. Zo kunnen we bijvoorbeeld componisten selecteren op basis van een deel van hun naam:

```
select naam   componist
from   Componist
where  naam like '%ar%';
```

Als resultaat krijgen we iedereen met 'ar' in de naam:

```
COMPONIST
==============
Charlie Parker
W.A. Mozart
Karl Schumann
```

Formaatstring %
_

De string met het procentteken heet een *formaatstring*. Het procentteken is een *wildcard* (vergelijk de 'joker' in het kaartspel) voor een willekeurige tekenrij. Er is ook een wildcard, de underscore, voor één teken: where naam like '_ar%' selecteert de componisten waarvan de tweede letter van de naam een 'a' is en de derde een 'r'.

6.5 LOGISCHE EXPRESSIES

Logische expressie
Logische waarde
Logische operator

De selectievoorwaarde (achter `where`) is een *logische expressie* die een *logische waarde* oplevert: *true*, *false* of *unknown*. In leereenheid 2 hebben we gezien hoe met de *logische operatoren* `and`, `or` en `not` samengestelde expressies kunnen worden gevormd volgens de regels van de driewaardige logica.

Hier volgen enkele voorbeelden:

Alle klassieke stukken van na 1980:

```
select *
from   Stuk
where  genre = 'klassiek' and jaar > 1980;
```

Alle klassieke stukken en ook alle stukken van na 1980:

```
select *
from   Stuk
where  genre = 'klassiek' or jaar > 1980;
```

Merk op dat in de vraagstelling 'en' staat en dat dit in de conditie een `or` wordt.

Alle niet-klassieke stukken:

```
select *
from   Stuk
where  not(genre = 'klassiek');
```

Een expressie met `between` ... `and` kan worden herschreven met gebruik van de logische operator `and`:

```
select nr, speelduur
from   Stuk
where  speelduur >= 5 and speelduur <= 8;
```

De selectieconditie hierin is equivalent met `speelduur between 5 and 8`.

6.6 PRIORITEIT VAN OPERATOREN

Prioriteit van operatoren

Gecombineerd gebruik van logische operatoren maakt dat we (net als bij bijvoorbeeld optellen en vermenigvuldigen) rekening moeten houden met de *prioriteit* (voorrangsregels) van de operatoren. De regel is dat `not` voorrang heeft op `and`, en `and` weer voorrang heeft op `or`.

De logische operatoren kunnen ook met andersoortige operatoren om de voorrang strijden. Tabel 4.2 geeft een overzicht van de prioriteitsregels.

TABEL 4.2 Prioriteit van operatoren

prioriteit	*operatoren*
1	||
2	+, – (unair)
3	*, /
4	+, – (binair)
5	=, <>, >, <. >=, <= in, like, is null, is not null, between ... and, not between ... and
6	not
7	and
8	or

Operatoren met prioriteit 1 hebben voorrang op alle andere; hoe hoger het nummer, des te lager de prioriteit.

VOORBEELD 4.10

```
select *
from   Stuk
where  not genre = 'klassiek' and niveau = 'A';
```

geeft alle niet-klassieke stukken van niveau A. In plaats van: `not genre = 'klassiek'` kan `genre <> 'klassiek'` natuurlijk ook.

```
select *
from   Stuk
where  genre = 'pop' or niveau = 'A' and speelduur <= 5;
```

geeft alle popstukken en daarnaast alle korte stukjes (maximaal 5 minuten) van niveau A.

Willen we een andere prioriteitsvolgorde dan overeenkomt met de voorrangsregels, dan moeten we net als bij rekenen haakjes gebruiken:

```
select *
from   Stuk
where  not(genre = 'klassiek' and niveau = 'A');
```

geeft alle stukken, behalve de klassieke stukken die niveau A hebben of die geen niveaucode hebben. Ook wanneer haakjes niet strikt nodig zijn, kan het voor de leesbaarheid nuttig zijn ze toe te voegen.

6.7 DE 'IS ELEMENT VAN'-OPERATOR IN

'Is-element-van'-operator `in`

Een `select`-query met `or` kan soms worden vereenvoudigd door het gebruik van de 'is element van'-operator `in`.

VOORBEELD 4.11

```
select *
from   Stuk
where  genre = 'klassiek' or genre = 'jazz' or genre = 'house';
```

kan worden vereenvoudigd tot:

```
select *
from   Stuk
where  genre in ('klassiek', 'jazz', 'house');
```

De tweede operand van `in` is een verzameling, de eerste is een element. De selectievoorwaarde geeft voor elke rij een *true* of een *false*, al naar gelang de waarde van genre tot de opgesomde verzameling behoort.

OPGAVE 4.8

Welke stukken (geef stuknummer en jaar) zijn geschreven in 1990, 1996 of 1998? Geef twee verschillende oplossingen.

6.8 SELECTEREN OP NULLS

Selecteren op nulls

Ook op nulls kan worden gecontroleerd binnen een selectievoorwaarde. In de volgende query worden alle originele stukken geselecteerd:

```
select *
from   Stuk
where  origineel is null;   -- alle originele stukken
```

Immers: de originele stukken zijn de niet-bewerkte stukken, dus de stukken waarvan origineel niet is ingevuld. Let op het gebruik van `is`: met '=' zouden we geen *true* maar *unknown* als resultaatwaarde krijgen bij een niet-ingevuld origineel.

Selecteren op not null

Bewerkingen zijn de stukken waarbij origineel juist wel is ingevuld, dus 'not null' is:

```
select *
from   Stuk
where  origineel is not null;   -- alle bewerkingen
```

Merk op dat '`is not null`' weer een staaltje is van 'Engels SQL'. Explicieter zou zijn: `not origineel is null`, of voor alle duidelijkheid met haakjes: `not(origineel is null)`.

OPGAVE 4.9

Speelstukken zijn te herkennen aan een ingevuld niveau, bewerkingen aan een ingevuld origineel.

a Geef van alle speelstukken: stuknummer, componist, titel en niveau.
b Geef de stuknummers van alle originele speelstukken.

7 Ordening: order by

De rijen in een relationele tabel zijn volgens de theorie ongeordend. Een tabel is immers een *verzameling* rijen en een verzameling kent geen volgorde. Willen we in het resultaat een bepaalde volgorde afdwingen, dan moeten we aan het einde van het `select`-statement een extra clausule opnemen: `order by`.

7.1 KLIMMEND EN DALEND ORDENEN

Klimmend ordenen

VOORBEELD 4.12

Het volgende `select`-statement laat een aantal gegevens van klassieke muziekstukken afdrukken op volgorde van niveau:

```
select   nr, genre, niveau, round(speelduur)
from     Stuk
where    genre = 'klassiek'
order by niveau;
```

Resultaat:

```
         NR GENRE      NIVEAU   ROUND
============ ========== ====== =======
          8 klassiek   <null>       4
         10 klassiek   <null>  <null>
         14 klassiek   A            4
          3 klassiek   B            5
          5 klassiek   B            5
          9 klassiek   B            4
```

`asc`, `desc`

Standaard wordt klimmend ('ascending') geordend. Desgewenst kunnen we dit expliciet in het statement opnemen: ... `order by` speelduur `asc`. Dalend ordenen ('descending') kan ook:

Dalend ordenen

VOORBEELD 4.13

```
select   nr, genre, niveau, round(speelduur)
from     Stuk
where    genre = 'klassiek'
order by niveau desc;
```

Resultaat:

```
         NR GENRE      NIVEAU   ROUND
============ ========== ====== =======
          9 klassiek   B            4
          3 klassiek   B            5
          5 klassiek   B            5
         14 klassiek   A            4
          8 klassiek   <null>       4
         10 klassiek   <null>  <null>
```

Null in ordening

Merk op dat bij klimmend ordenen de rijen met een null in de ordeningskolom vooraan staan en bij dalend ordenen achteraan. Dit kan per dialect verschillen.

7.2 ORDENEN OP EXPRESSIE

Het volgende voorbeeld illustreert dat een `order by`-clausule een berekende expressie mag bevatten. Hiertoe kan de expressie zelf worden opgenomen of een kolomalias.

VOORBEELD 4.14 (ordenen op berekende expressie)

Hetzelfde overzicht als van voorbeeld 4.12, maar nu geordend op de (afgeronde) speelduur:

```
select   nr, genre, niveau, round(speelduur)
from     Stuk
where    genre = 'klassiek'
order by round(speelduur);
```

Alternatief met kolomalias:

```
select   nr, genre, niveau, round(speelduur) speelduur
from     Stuk
where    genre = 'klassiek'
order by speelduur;
```

De 'speelduur' in de `order by`-clausule verwijst nu naar de kolomalias, niet naar de (gelijknamige) kolomnaam.

Bij al deze voorbeelden werd geordend op een kolom van de `select`-lijst. Hoewel dat bijna altijd zal zijn wat we willen, is het ook toegestaan om te ordenen op een willekeurige, andere expressie.

7.3 VERFIJND ORDENEN

Verfijnd ordenen

Bij ordening op één kenmerk komen rijen waarvoor dat kenmerk dezelfde waarde heeft, weliswaar opeenvolgend in het resultaat, maar onderling in willekeurige volgorde. Het is mogelijk binnen elk groepje rijen te ordenen op een extra kenmerk. We spreken van *verfijnd ordenen*. Het is mogelijk verfijnd te ordenen op een willekeurig aantal niveaus diep.

VOORBEELD 4.15

Het volgende statement geeft van elk muziekstuk een aantal kenmerken, alfabetisch geordend op genre en per genre van moeilijk naar makkelijk.

```
select   nr, genre, niveau, speelduur, jaar
from     Stuk
order by genre asc, niveau desc;
```

Resultaat:

```
        NR GENRE      NIVEAU SPEELDUUR    JAAR
========== ========== ====== ========= =======
         2 jazz       B            4.0    1988
        13 jazz       B            8.0    1997
        15 jazz       A            4.0    1998
         1 jazz       <null>       4.5    1954
         9 klassiek   B            3.5    1990
         3 klassiek   B            4.5    1953
         5 klassiek   B            5.0    1979
        14 klassiek   A            4.3    1998
         8 klassiek   <null>       4.0    1786
        10 klassiek   <null>    <null>    1791
        12 pop        A            6.0    1996
```

De vermelding `asc` mag ook hier achterwege blijven.

Ordening via kolomvolgnummer

Syntactische varianten
Naar bij name genoemde kolommen of expressies in de `select`-lijst, mag in de `order by`-clausule gerefereerd worden via hun volgnummers in de lijst. De `order by`-clausule in voorbeeld 4.15 mag daarom ook als volgt luiden:

```
order by 2 asc, 3 desc
```

Ooit was referentie via volgnummers de enige manier om op een berekende expressie te ordenen. Dat kan nog steeds gelden voor sommige dialecten. Wanneer, zoals in Firebird, de `order by`-clausule een berekende expressie of een kolomalias mag bevatten, ligt het gebruik van volgnummers minder voor de hand.

OPGAVE 4.10
Produceer een overzicht van alle speelstukken (stukken met een niveau). Geef van elk stuk het stuknummer en de genre-niveau-combinatie. Orden het overzicht alfabetisch op genre en per genre op niveau, van moeilijk naar makkelijk. Als volgt:

```
      NR GENRE_EN_NIVEAU
======== ==============
       2 jazz/B
      13 jazz/B
      15 jazz/A
       3 klassiek/B
       5 klassiek/B
       9 klassiek/B
      14 klassiek/A
      12 pop/A
```

8 Verzamelingsoperatoren

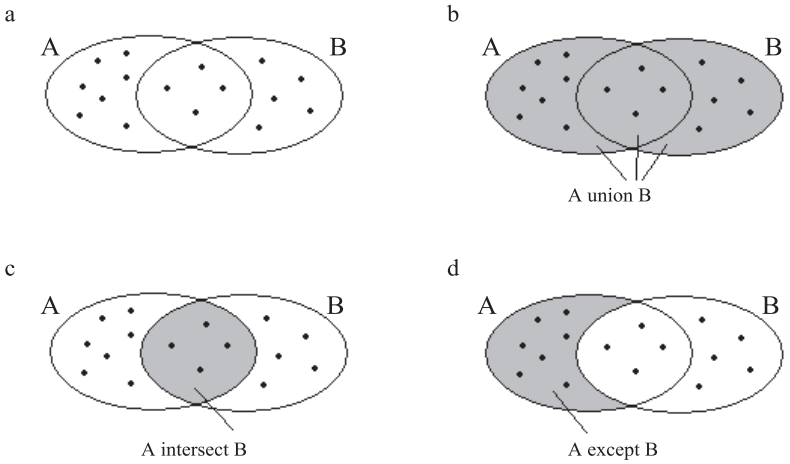
Verzamelings-operator

Een tabel is een verzameling rijen. Wanneer twee tabellen hetzelfde aantal kolommen hebben met twee-aan-twee vergelijkbare datatypen (numeriek, alfanumeriek of datum), kunnen we op verschillende manieren uit hun rijen één nieuwe tabel maken. Hiervoor bestaan de volgende binaire *verzamelingsoperatoren*:

– de operator `union` voegt de tabellen rijgewijs samen
– de operator `intersect` neemt de gemeenschappelijke rijen
– operator `except` neemt de eerste tabel en schrapt daarin de rijen van de tweede tabel.

Deze operatoren werken écht relationeel: gelijke rijen in hun tabel-operanden leiden tot één resultaatrij. Ze verdienen daarom de naam *relationele operator*. In de wiskunde zijn deze voor willekeurige verzamelingen gedefinieerd, reden waarom ze ook in de relationele theorie met de algemenere term *verzamelingsoperator* worden aangeduid. Hun namen in de verzamelingenleer luiden achtereenvolgens: *vereniging*, *doorsnede* en *verschil*.

Figuur 4.6 illustreert hun werking:



FIGUUR 4.6 De relationele operatoren `union`, `intersect` en `except`

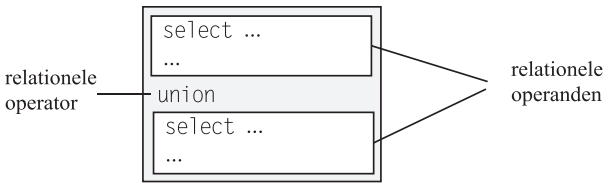
Figuur a representeert twee tabellen, A en B, met respectievelijk elf en negen rijen, elke rij gesymboliseerd door een puntje. A bevat vier rijen die precies gelijk zijn aan vier rijen van B; dit wordt weergegeven door de vier puntjes in het overlappende gedeelte. Fysiek zijn er dus twintig rijen, maar slechts twaalf hiervan zijn onderling verschillend. Figuur b representeert de tabel A `union` B, met zestien rijen. Figuur c symboliseert A `intersect` B, met de vier rijen die zowel in A als in B voorkomen. Tot slot bevat figuur d een weergave van A `except` B, met de zeven rijen die wel in A, maar niet in B voorkomen.

De operator `union` is in de praktijk de belangrijkste. Het is de enige van de drie die in Firebird beschikbaar is. Het gebruik ervan wordt in de volgende paragraaf aan de hand van voorbeelden toegelicht. In de paragraaf erna geven we ook een enkel voorbeeld van de operatoren `intersect` en `except`.

8.1 VERENIGING

Operator union

Met de verenigingsoperator `union` kunnen twee tabellen tot één tabel worden samengevoegd. Die resultaattabel omvat alle rijen die tot de ene tabel, tot de andere tabel of tot beide behoren.
Rijen die in beide tabellen voorkomen, worden maar één keer in de resultaattabel opgenomen. De operator `union` is dus een echte relationele operator, zie ook figuur 4.7.



FIGUUR 4.7 Union: een relationele operator

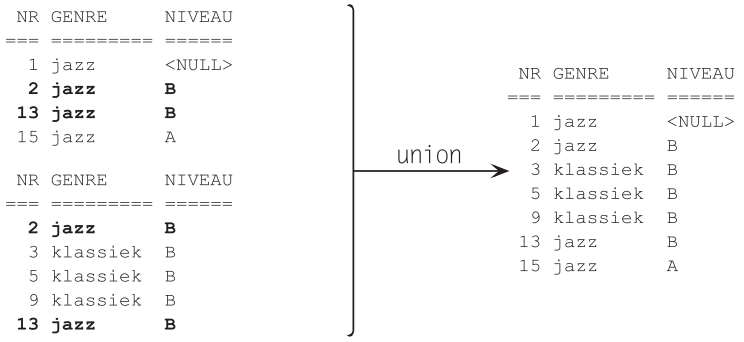
VOORBEELD 4.16
(elementair, maar niet erg nuttig)

Gebruik de `union`-operator om een overzicht te geven van alle stukken die tot het genre 'jazz' behoren en/of niveau B hebben.

Oplossing:

```
select nr, genre, niveau
from   Stuk
where  genre = 'jazz'
union
select nr, genre, niveau
from   Stuk
where  niveau = 'B';
```

We zien: twee `select`-expressies waarvan de resultaattabellen hetzelfde type kolommen hebben (zelfs afkomstig van dezelfde basistabel) en die via een `union`-operatie tot één tabel worden samengevoegd, zie figuur 4.8. Daarin staan linksboven alle jazz-stukken en linksonder alle stukken van niveau B. Stukken die zowel linksboven als linksonder voorkomen zijn vetgedrukt.



FIGUUR 4.8 `union`-operatie op deeltabellen van één tabel

Dat is wel simpel en misschien ook instructief, maar niet direct waar we op zaten te wachten. Hetzelfde resultaat kunnen we immers eenvoudiger bereiken met de logische operator `or`:

```
select nr, genre, niveau
from   Stuk
where  genre = 'jazz' or niveau = 'B';
```

Het volgende voorbeeld geeft een zinvoller toepassing.

VOORBEELD 4.17

Geef van elk stuk nummer en titel. Vermeld in een derde kolom of het stuk een origineel is of een bewerking en in het laatste geval ook waarvan het een bewerking is.

Oplossing:

```
-- de originelen
select nr, titel, 'origineel' origineel_of_bewerking
from   Stuk
where  origineel is null
union
-- de bewerkingen
select nr, titel, 'bewerking van ' || origineel
from   Stuk
where  origineel is not null;
```

Resultaat:

```
      NR TITEL            ORIGINEEL_OF_BEWERKING
======== ===============  ======================
       1 Blue bird        origineel
       2 Blue bird        bewerking van 1
       3 Air pour charmer origineel
       5 Lina             origineel
       8 Berceuse         origineel
       9 Cradle song      bewerking van 8
      10 Non piu andrai   origineel
      12 I'll never go    bewerking van 10
      13 Swinging Lina    bewerking van 5
      14 Little Lina      bewerking van 5
      15 Blue sky         bewerking van 1
```

Toelichting
– Er is maar één resultaattabel; een kolomalias is daarom alleen in de eerste `union`-operand zinvol.
– De ordening in de resultaattabel op nr wordt niet expliciet afgedwongen en is dus niet iets waar u op kunt rekenen. Om een bepaalde volgorde af te dwingen, is een `order by`-clausule nodig, zie verderop in paragraaf 8.3.

In voorbeeld 4.17 wordt getest op het al dan niet null zijn van de kolomwaarde origineel. Dit doet denken aan de `coalesce`-functie. Inderdaad kan de `union`-constructie worden vermeden door een `coalesce`-aanroep te gebruiken:

```
select  nr, titel, coalesce('bewerking van ' || origineel, 'origineel')
from    Stuk;
```

Hierin wordt expliciet gebruikt dat concatenatie met een null als resultaat een null geeft, iets wat niet voor elk SQL-dialect geldt. Ook dit voorbeeld is dus geen illustratie van de noodzaak van de `union`-operator. Overtuigender zou een voorbeeld zijn met twee `union`-operanden die op verschillende brontabellen zijn gebaseerd. In de praktijk komen we dit vooral tegen in slecht ontworpen databases met ‘multiple points of definition’; met bijvoorbeeld niet-gestandaardiseerde plaatsnamen die over verschillende tabellen zijn verspreid. Hoe beter de database is gemodelleerd, des te minder is de `union` nodig. Niettemin zullen we de `union` zo nu en dan gebruiken, als een nuttige aanvulling op ons repertoire.

8.2 DOORSNEDE EN VERSCHIL

Operatoren intersect en except

De verzamelingsoperator `intersect` geeft van twee tabellen de *doorsnede* (of *intersectie*), dat is de tabel bestaande uit alle gemeenschappelijke rijen. De verzamelingsoperator `except` geeft van twee tabellen hun *verschil*, dat zijn alle rijen die wel in de eerste maar niet in de tweede tabel voorkomen. Van deze laten we nu één voorbeeld volgen, de oplossing van een 'ouders zonder kind'-probleem.

VOORBEELD 4.18

Van welke genres is geen enkel stuk aanwezig?

Oplossing:

```
select naam
from   Genre
except
select genre
from   Stuk;
-- (geen Firebird)
```

Resultaat:

```
NAAM
---------
wereldmuziek
```

Niet-Firebird query's
Niet-Firebird query's zijn door ons zoveel mogelijk in Oracle uitgevoerd; hun resultaattabellen zijn herkenbaar aan een enkele stippellijn.

Zowel een `except`- als een `intersect`-expressie resulteren in een tabel die kleiner is dan of even groot als de eerste operand. De tweede operand impliceert in beide gevallen een beperking die aan de eerste operand wordt opgelegd. Daardoor zijn expressies van beide typen vrijwel altijd equivalent met de eerste operand waarin de `where`-clausule is uitgebreid met een aanvullende conditie.
Dat geldt ook voor de query van voorbeeld 4.18, die equivalent is met een query met de volgende structuur:

```
select naam
from   Genre
where  ...
```

Alternatieven voor intersect en except

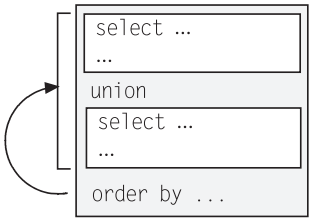
In gewoon Nederlands komt de ontbrekende conditie neer op: 'bij deze ouderrij bestaat geen enkele corresponderende kindrij in de tabel Stuk'. We moeten echter nog even tot leereenheid 7 'Subselects en views' wachten om dit in SQL te kunnen vertalen. Al met al blijkt hieruit wel waarom het in de praktijk geen probleem is wanneer een SQL-dialect geen `intersect` of `except` kent.

8.3 VERZAMELINGSEXPRESSIES EN ORDENING

Verzameling-expressies en order by

Alleen aan het einde van een verzamelingexpressie mag een `order by`-clausule worden toegevoegd. Deze ordent het eindresultaat. Zie figuur 4.9, waarin dit wordt geïllustreerd voor de `union`. Dat de `order by` alleen aan het einde mag, is logisch: ordening is immers geen relationele operatie, alleen het zichtbare eindresultaat is iets dat kan worden geordend.

Binnen de `order by`-clausule van een union mag in Firebird een kolom of expressie alleen met zijn volgnummer worden aangegeven.



FIGUUR 4.9 `order by` werkt op de hele `union`-expressie

OPGAVE 4.11
Los op met `union`: geef van alle stukken componist, titel en niveau.
Wanneer een stuk geen speelstuk is (en dus geen niveaucode heeft), moet in plaats van het niveau de tekst 'geen speelstuk' worden afgedrukt.
Orden het overzicht op niveau (de niet-speelstukken achteraan), per niveau op componist en daarbinnen alfabetisch op titel.
(In opgave 4.5 werd dit zelfde probleem – afgezien van de ordening – opgelost met behulp van de null-vervangfunctie `coalesce`.)

SAMENVATTING

Paragraaf 2

Een `select`-statement is een relationele operator, werkend op een tabel als operand (de brontabel) en met een tabel als resultaatwaarde.
Een eenvoudige vorm ervan is de projectie op een deelverzameling van de kolommen. Bij een gegeneraliseerde vorm van de projectie mag de `select`-lijst ook constanten of berekende expressies bevatten.
Elementen uit de `select`-lijst kunnen een kolomalias krijgen, die is op te vatten als een gewone kolomnaam van de resultaattabel.
Door de `select`-lijst vooraf te laten gaan door `distinct`, worden dubbele rijen in de presentatie van de resultaattabel onderdrukt.

Paragraaf 3

Gegevens in één kolom zijn van een bepaald datatype. De SQL-standaard schrijft onder meer voor: `integer` (gehele getallen), `numeric` (algemenere numerieke waarden), `char` (tekst van vaste lengte), `varchar` (tekst van variabele lengte) en `date` (kalenderdata).
Bij tekst (alfanumerieke tekenrijen) kan een *character set* zijn gedefinieerd, die vastlegt hoe de intern-geheugenrepresentatie van tekens dient te worden vertaald naar een bepaalde verschijningsvorm van het teken, en omgekeerd. Veel character sets hebben de ASCII-set als deelverzameling (de eerste 128 tekens).
Via *typecasting* kan soms een waarde van het ene datatype in een waarde van het andere datatype worden geconverteerd.

Paragraaf 4

Een *operator* is een teken of samenstel van tekens waarmee een bewerking wordt gedefinieerd. Een operator werkt op een of meer *operanden*. De meeste operatoren zijn binair (werkend op twee operanden). Afhankelijk van het datatype van de operanden onderscheiden we numerieke operatoren, alfanumerieke operatoren, enzovoort.
Soms is expliciete typecasting nodig. Soms gebeurt dit automatisch.

Paragraaf 5

Een *functie* retourneert, net als een operator, een resultaatwaarde op de plaats waar zij wordt aangeroepen. Een functie heeft een naam en werkt op nul of meer *argumenten*, die in de aanroep tussen haakjes worden meegegeven. Afhankelijk van het datatype van de argumenten (soms van de geretourneerde waarde) spreekt men van numerieke functies, datumfuncties, enzovoort.
Een belangrijke mogelijkheid is zelf functies te definiëren: *user defined functions* (*udf*'s).

Paragraaf 6

Door een `select`-statement uit te breiden met een `where`-clausule met selectieconditie, beperken we de resultaattabel tot de rijen die aan de conditie voldoen: een selectie.
Een selectieconditie is een logische expressie van de driewaardige logica. Belangrijke operatoren om logische expressies mee te vormen zijn de vergelijkingsoperatoren (=, <>, >, <, >=, <=), de ternaire operator `between ... and`, de tekstvergelijkingsoperator `like`, de 'is element van'-operator `in`. Van bijzonder belang zijn de operatoren `is null` en `is not null` voor het selecteren op lege of juist niet-lege cellen.
Om samengestelde logische expressies te vormen dienen de logische operatoren `and`, `or` en `not`. Hierbij is het van belang te letten op de prioriteit van de operatoren en zo nodig haakjes te gebruiken.

Paragraaf 7

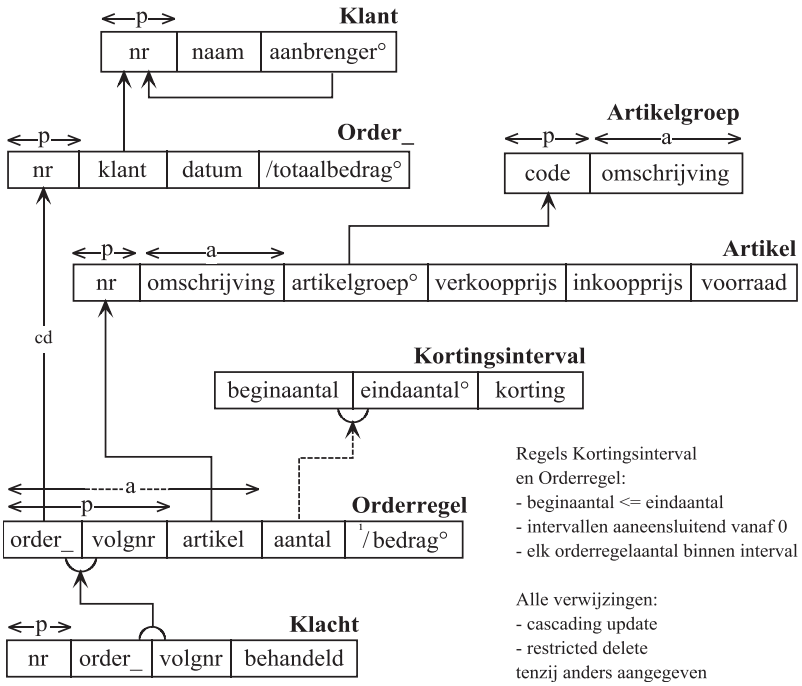
Door een `select`-statement uit te breiden met een `order by`-clausule kan een ordening (klimmend of dalend) van de resultaattabel worden afgedwongen op één van de elementen van de `select`-lijst. Ook kan *verfijnd* worden geordend op een combinatie van elementen. Het gaat hierbij om *presentatie*; vanuit relationeel oogpunt is elke tabel ongeordend.

Paragraaf 8

De verzamelingsoperator `union` werkt op twee tabeloperanden, waarvan de kolommen twee aan twee van vergelijkbaar datatype moeten zijn (bijvoorbeeld: beide alfanumeriek). De operator `union` komt overeen met de verenigingsoperator uit de verzamelingenleer. De noodzaak `union` te gebruiken kan duiden op een slecht ontworpen database.
De SQL-standaard kent ook de verzamelingsoperatoren `intersect` en `except`, die overeenkomen met de doorsnede- en verschiloperator uit de verzamelingenleer. Expressies met `intersect` en `except` kunnen vrijwel altijd worden herschreven tot één `select`-expressie met een geschikte selectieconditie.

ZELFTOETS

De volgende opgaven hebben betrekking op de voorbeelddatabase Orderdatabase. We herhalen hier het strokendiagram; de voorbeeldpopulatie vindt u achterin.



FIGUUR 4.10 Strokendiagram Orderdatabase

1 Geef een overzicht van alle orderregels waarover een of meer klachten zijn binnengekomen. Het is voldoende om van die orderregels alleen het ordernummer en het volgnummer op te nemen.

2 Geef een artikeloverzicht, met prijzen inclusief btw, afgerond op eurocenten, als volgt:

```
PRIJSLIJST
==========================
Artikel 107 kost 2.62 euro
Artikel 180 kost 30.94 euro
...
```

Neem aan dat de opgeslagen verkoopprijzen exclusief btw zijn. Ga uit van een 'hard' in te voeren btw-percentage (19%).

3 Geef alle artikelomschrijvingen met een beginhoofdletter en verder in kleine letters.
Aanwijzing
Gebruik hierbij de functies `upper`, `lower` en `substring` (zie bijlage 1 in cursusdeel 2).

4 Geef een overzicht van alle artikelen, met hun artikelnummer en omschrijving, geordend op artikelnummer. Geef in een derde kolom de artikelen die niet tot een artikelgroep behoren, de vermelding ‘los artikel’.

5 Geef een overzicht van alle artikelen, met hun artikelnummer, omschrijving en – indien aanwezig – artikelgroep. Artikelen die niet tot een artikelgroep behoren, moeten in plaats daarvan de vermelding ‘los artikel’ krijgen. Orden het overzicht op artikelgroep (met aan het einde de losse artikelen) en bij gelijke artikelgroep op artikelnummer. U mag hierbij aannemen dat artikelgroepcodes met een hoofdletter beginnen.

TERUGKOPPELING

1 Uitwerking van de opgaven

4.1 a Het Muziekpakhuis.

b Geen uitwerking.

c Alleen met betrekking tot de gegeven populatie, waarin *toevallig* één stuk met de titel 'Non piu andrai' voorkomt, mogen we spreken over hét stuk met die titel. De structuur laat toe dat er meerdere stukken zijn met dezelfde titel. Vanuit de structuur mogen we dus hooguit spreken over *een* stuk met een bepaalde titel.

d Een bewerking, want in de kolom origineel van de betreffende rij staat dat het origineel van dit stuk het stuk met nr 5 is.

e Het stuk 'Little Lina' van Sofie Bergeijk is een bewerking van haar eigen stuk 'Lina' (stuk 5 en stuk 14 hebben beide componist 5).

f De bewerkingen met stuknummers 9, 12, 13, 14 en 15 hebben een andere titel dan hun respectievelijke originelen.

4.2 SQL-statement:

```
select distinct instrument, toonhoogte
from   Bezettingsregel;
```

4.3 SQL-statement:

```
select toonhoogte || naam    instrument
from   Instrument;
```

Bij instrumenten 'zonder toonhoogte' zoals de piano wordt een lege string geconcateneerd met de naam van het instrument. De hier gegeven oplossing is dialectspecifiek, ook al in het databaseontwerp. Bij andere dialecten moet rekening worden gehouden met de volgende probleempunten (zie leereenheid 2):
– Codd-relationaliteit
– het onderscheid tussen null en de lege string
– concatenatie waarbij één van de operanden een null is.

NB We hebben de kolom van de resultaattabel de naam 'instrument' gegeven (in plaats van 'concatenation', die we anders gekregen zouden hebben) door middel van een kolomalias. Dit is niet verplicht en werd ook niet gevraagd, maar het mag natuurlijk wel.

4.4 SQL-statement:

```
select current_date + 1000
from   Rdb$database;
```

4.5 SQL-statement:

```
select componist, titel, coalesce(niveau, 'geen speelstuk') niveau
from   Stuk;
```

4.6 De volgende `select`-expressie:

```
select cast('1582-10-15' as date) - cast('1582-10-04' as date)
from   Rdb$database;
```

geeft als antwoord: 11. Conclusie: de SQL-datumrekenkunde (in elk geval van Firebird) houdt geen rekening met de datumsprong van 1582.

NB: Een typecast kan ook iets korter (en misschien iets minder goed leesbaar) worden opgeschreven zonder expliciet de `cast`-functie aan te roepen:

```
select date '1582-10-15' - date '1582-10-04'
from   Rdb$database;
```

4.7 Met behulp van de functie `iif` kunnen we de genoemde vergelijkingen laten uitrekenen, en bijvoorbeeld 'ja' laten afdrukken als de vergelijking waar is, en 'nee' anders. Voor het vergelijken van datumwaarden krijgen we dan bijvoorbeeld:

```
select iif(cast('31-dec-1999' as date) < cast('01-jan-2000' as date),
           'ja', 'nee')
from   Rdb$database;
```

met als uitvoer:

```
CASE
======
ja
```

en voor het vergelijken van de strings:

```
select iif('31-dec-1999' < '01-jan-2000', 'ja', 'nee')
from   Rdb$database;
```

met als uitvoer

```
CASE
======
nee
```

4.8 Oplossing met `or`:

```
select nr, jaar
from   Stuk
where  jaar = 1990 or jaar = 1996 or jaar = 1998;
```

Oplossing met `in`:

```
select nr, jaar
from   Stuk
where  jaar in (1990, 1996, 1998);
```

4.9 a SQL-statement:

```
select nr, componist, titel, niveau
from   Stuk
where  niveau is not null;
```

b SQL-statement:

```
select nr
from   Stuk
where  niveau is not null       -- speelstuk
       and origineel is null;   -- origineel stuk
```

4.10 SQL-statement:

```
select   nr, genre || '/' || niveau   genre_en_niveau
from     Stuk
where    niveau is not null   -- speelstuk
order by genre asc, niveau desc;
```

4.11 SQL-statement:

```
select componist, titel, niveau
from   Stuk
where  niveau is not null
union
select componist, titel, 'geen speelstuk'
from   Stuk
where  niveau is null
order  by 3, 1, 2;
```

2 Uitwerking van de zelftoets

1 SQL-statement:

```
select distinct order_, volgnr
from   Klacht;
```

Opmerking
Bij de huidige populatie lijkt de `distinct` misschien overbodig, maar dat is hij niet! De combinatie (order_, volgnr) hoeft niet uniek te zijn in de tabel Klacht, dus orderregels met meer dan één klacht kunnen vaker voorkomen. De vraagstelling maakt geen onderscheid tussen orderregels met één klacht en orderregels met meer dan één klacht, dus we willen iedere orderregel met één of meer klachten maar eenmaal in het resultaat zien.

2 SQL-statement:

```
select 'Artikel ' || nr || ' kost ' || round(verkoopprijs * 1.19, 2)
       || ' euro'  prijslijst
from   Artikel;
```

3 SQL-statement:

```
select upper(substring(omschrijving from 1 for 1)) ||
       lower(substring(omschrijving from 2))  artikel
from   Artikel;
```

4 Er wordt niet gespecificeerd wat er in de derde kolom moet staan als het betreffende artikel wél tot een artikelgroep behoort: waarschijnlijk niet de artikelgroepcode (gezien de vraagstelling), maar wat dan? Null? De lege string? Een spatie? Los van deze keuze zijn er verschillende manieren om deze opgave op te lossen, waarvan we er twee laten zien.

Als eerste een oplossing met een union:

```
select   nr, omschrijving, null
from     Artikel
where    artikelgroep is not null
union
select   nr, omschrijving, 'los artikel'
from     Artikel
where    artikelgroep is null
order by 1;
```

Opmerkingen:
– in de laatste regel is `order by nr` niet toegestaan
– we hebben nu voor de null gekozen, maar dat had net zo goed de lege string of de spatie kunnen zijn.

De tweede oplossing maakt gebruik van de functie `iif` en de lege string (maar dat had net zo goed de null of een spatie kunnen zijn):

```
select nr, omschrijving, iif(artikelgroep is null, 'los artikel', '')
from   Artikel;
```

5 Ook dit kan met een `union`. Met de null-vervangfunctie `coalesce` kan het echter heel wat simpeler:

```
select   nr, omschrijving, coalesce(artikelgroep, 'los artikel')
from     Artikel
order by 3, 1;
```

In de `order by`-clausule mogen ook de expressies van de `select`-clausule worden gebruikt. Ook kunt u de `coalesce`-expressie een alias geven en deze hier opnemen.