Relationele databases en SQL, 3e druk

Uitwerkingen van opgaven (versie 20140603)

Uitwerkingen hoofdstuk 1

1.1 $0.15 \times 1600 + 0.5 \times 40 + 0.3 \times 336 + 10 \times 4 + 1 \times 30$ geeft (na afronding): 431.

1.2 De volgende rijen komen erbij:

in Gerecht: ('Reijnders Roem', 86, 12, 'Snijd het fruit in stukjes, vermeng alles met de pernod en laat dit een uurtje staan alvorens het op te dienen')

in Product: ('meloen', 'stuks', 150)

in Ingredient: ('Reijnders Roem', 'kiwano', 0.5)
('Reijnders Roem', 'mango', 0.2)
('Reijnders Roem', 'meloen', 0.1)
('Reijnders Roem', 'pernod', 1)

1.3a De drie Ingredient-rijen van Mango Plus Plus verwijderen en daarna de Gerecht-rij van Mango Plus Plus. (In die volgorde, anders wijst het gerecht Mango Plus Plus in Ingredient tijdelijk niet naar een bestaande rij in Gerecht.)

b Verwijderen van de banaanrij in Product. Er zijn geen gevolgen voor andere tabellen of belemmeringen, omdat geen enkel gerecht banaan bevat.

c Product moet er een kiwirij bij krijgen: ('kiwi', 'stuks', 30), Ingredient krijgt een nieuwe rij ('Mango Plus Plus', 'kiwi', 1) en in Gerecht moet energiePP van Mango Plus Plus worden herberekend.

d Aan Eenheid moet een rij voor 'koffielepel' worden toegevoegd. In Product moet voor pernod de waarde van eenheid worden veranderd in 'koffielepel' en moet de waarde van energiePE door een factor 2.5 worden gedeeld. In Ingredient moet hoeveelheidPP in de rij van (Glace Terrace, pernod) juist met 2.5 worden vermenigvuldigd. In Gerecht verandert niets.

1.4 Na verwijderen van de mangorij in Product zou de (Mango Plus Plus, mango)-rij in Ingredient een 'loze' verwijzing bevatten, naar een Product-rij die er niet meer is. Dit zou die hele rij betekenisloos maken en is daarom verboden.
Deze opgave lijkt op opgave 3a, maar er is een belangrijk verschil: gerechten staan nooit los van hun ingrediënten, maar producten hebben een belang op zichzelf. Daarom is het heel normaal een Gerecht-rij te verwijderen met Ingredient-rijen en al, maar zou het onaanvaardbaar zijn om zomaar een Product-rij te verwijderen met alle bijbehorende Ingredient-rijen. Elk recept met dat product als ingrediënt zou daardoor veranderen.

61.5g De '8' op zichzelf heeft geen informatiewaarde omdat er geen atomaire zin bestaat waarin alleen die '8' als invulwaarde voorkomt. De kleinste atomaire zin met de '8' erin bevat ook een gerecht: 'het gerecht Mango Plus Plus heeft een bereidingstijd van 8 minuten'.

U zou kunnen tegenwerpen dat de '8' op zichzelf ook wat betekent: 'er bestaat een gerecht met een bereidingstijd van 8 minuten'. Dit beschouwen we echter niet als een 'erkende' zin. We kijken per tabel alléén naar bepaalde 'dingen' en hun eigenschappen. In dit geval is Mango Plus Plus het ding en een bereidingstijd van 8 minuten een eigenschap daarvan.

1.6a De enige plaats om informatie over artikelen op te nemen, is in de kolommen artikelnr, omschrijving en prijs van de subtabel 'orderregels'. Een heel gekunstelde oplossing zou dan zijn: deze kolommen vullen en de rest leeg laten, zodat we als het ware artikelen opslaan in een 'lege order'. Zo'n oplossing is niet alleen gekunsteld, maar echt fout in het licht van de 'normale' semantiek van de tabel. Alle gegevens zeggen immers direct of indirect iets over een order.
Voor een zeer nauwkeurig antwoord op de vraag zouden we moeten beschikken over een expliciete semantiek van de tabel, bijvoorbeeld in de vorm van een verwoording van alle betekenissen door middel van atomaire zinnen.

b en c Zie figuur 1.16. Omdat de gegevens zijn gestructureerd vanuit het perspectief van de artikelen, kunnen alle artikelen op een natuurlijke manier worden toegevoegd, ook als ze op geen enkele order voorkomen. De vetgedrukte rij illustreert dit.

Artikel

nr	omschrijving	prijs	orderregels					
			ordernr	klantnr	klantnaam	datum	aantal	bedrag
107	zwenkmoer	2.20	5773	1234	Cupido	20-okt-2011	10	22.00
180	zwalik	26.00	5774	1447	Roos	20-okt-2011	1	26.00
			5793	1234	Cupido	22-okt-2011	2	52.00
351	slevel	15.80	5773	1234	Cupido	20-okt-2011	2	31.60
			5774	1447	Roos	20-okt-2011	2	31.60
449	wigbek	13.50	5773	1234	Cupido	20-okt-2011	3	41.50
			5793	1234	Cupido	22-okt-2011	1	13.50
371	**draaikloon**	**175.00**						

Figuur 1.16 Omkering van de herhalende groep van figuur 1.15 vanuit artikelperspectief, met extra artikelrij

Gegevens vanuit klantperspectief

Figuur 1.17 geeft het beloofde extraatje: de gegevens gestructureerd vanuit het perspectief van de klanten. Omdat een klant meerdere orders kan hebben geplaatst en een order meerdere orderregels kan hebben, bevat elke orders-subtabel op zijn beurt een orderregels-subtabel. We spreken van een herhalende groep die is 'genest' binnen een andere herhalende groep.

Klant

nr	klantnaam	orders							
		ordernr	datum	totaalbedrag	orderregels				
					artikelnr	omschrijving	prijs	aantal	bedrag
1234	Cupido	5773	20-okt-2011	94.10	351	slevel	15.80	2	31.60
					107	zwenkmoer	2.20	10	22.00
					449	wigbek	13.50	3	41.50
		5793	22-okt-2011	65.50	449	wigbek	13.50	1	13.50
					180	zwalik	26.00	2	52.00
1447	Roos	5774	20-okt-2011	57.60	351	slevel	15.80	2	31.60
					180	zwalik	26.00	1	26.00

Figuur 1.17 De gegevens vanuit klantperspectief, met 'geneste' herhalende groep

1.7a Zie figuur 1.17.

Order

nr	klantnr	klantnaam	datum	totaalbedrag
5773	1234	Cupido	20-okt-2011	94.10
5774	1447	Roos	20-okt-2011	57.60
5793	1234	Cupido	22-okt-2011	65.50

Orderregel

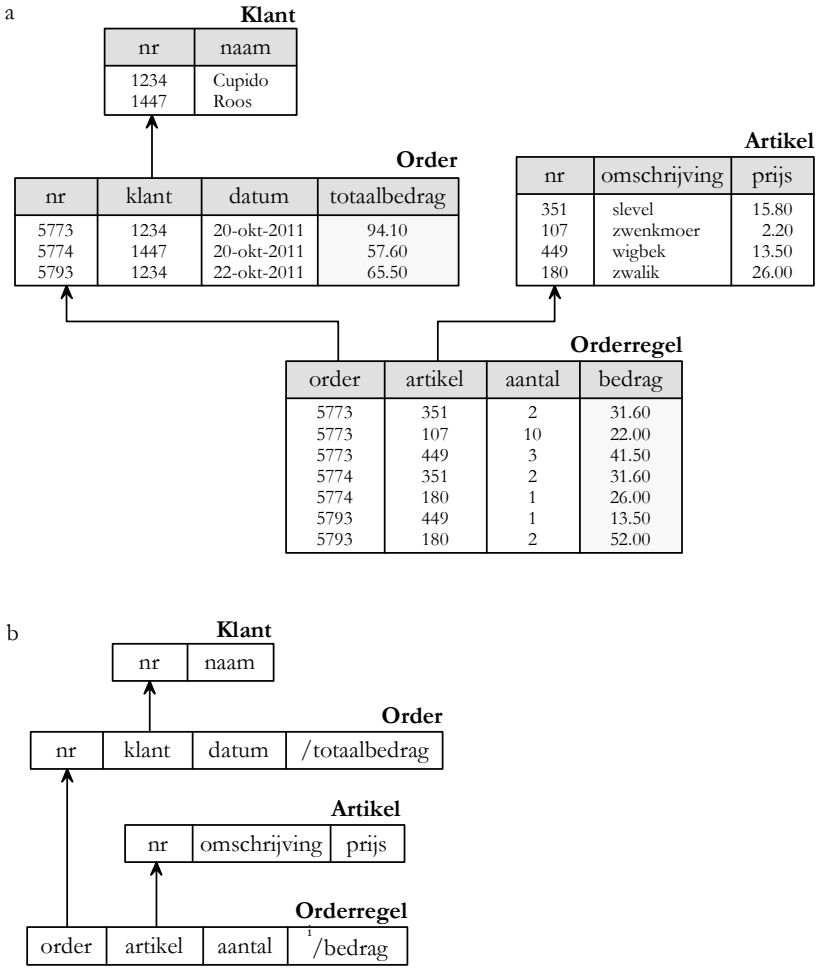
order	artikelnr	omschrijving	prijs	aantal	bedrag
5773	351	slevel	15.80	2	31.60
5773	107	zwenkmoer	2.20	10	22.00
5773	449	wigbek	13.50	3	41.50
5774	351	slevel	15.80	2	31.60
5774	180	zwalik	26.00	1	26.00
5793	449	wigbek	13.50	1	13.50
5793	180	zwalik	26.00	2	52.00

Figuur 1.18 Resultaattabellen na afsplitsen van herhalende groep

b Een klantnaamvermelding van elke klant die vaker voorkomt, is redundant. Ook de artikelomschrijving en de prijs van elk artikel dat vaker voorkomt, zijn redundant.

c Elimineren van beide typen redundantie gebeurt door zowel klanten als artikelen (allebei 'soorten dingen') hun eigen tabel te geven, waarin hun kenmerken eenmalig en dus eenduidig worden vastgelegd. Vanuit de (afgeslankte) al bestaande tabellen wordt daarnaar verwezen. In figuur 1.19 ziet u het resultaat in de vorm van een populatiediagram (a) en een strokendiagram (b).

Merk op dat de kolommen Orderregel.bedrag en Order.totaalbedrag als afleidbaar zijn gemarkeerd. (Zie de uitwerking van opgave 1.8 voor uitleg over /bedrag in het strokendiagram.)



Figuur 1.19 Eindresultaat afsplitsing: populatie (a) en strokendiagram (b)

Naamgeving
In de tussenresultaten, waarin nog niet elk soort ding zijn eigen tabel had, was een strakke naamgeving nog niet goed mogelijk. In het eindresultaat kan dat heel goed. Volgens onze *huisstijl* geven we een identificerende nummerkolom veelal de kolomnaam 'nr': Klant.nr, Order.nr en Artikel.nr. In een verwijzende kolom leiden we de kolomnaam meestal af uit de naam van de tabel waarnaar wordt verwezen: Order.klant, Orderregel.order en Orderregel.artikel.

NB: de verwijzing staat technisch gezien los van de naamgeving. Het gaat puur om begrijpelijkheid voor de menselijke lezer, waarbij een keuze is gemaakt voor een bepaalde (huis)stijl.

1.8 Al worden de orderregelbedragen automatisch berekend, toch moeten ze in de database worden opgeslagen. Ze zijn namelijk alleen maar afleidbaar gedurende de ordertransactie, dat wil zeggen: zolang de nieuwe order en haar orderregels worden ingevoerd of aangepast, en tot het moment dat deze definitief worden vastgelegd. We spreken van *initieel afleidbaar*. Dit wordt aangegeven met de "i" in het strokendiagram. Na een prijswijziging van een artikel op een order is het gedaan met de afleidbaarheid. De bedragen worden dus eenmalig automatisch berekend en vervolgens als onveranderlijke waarden opgeslagen. Zo wordt voorkomen dat bij latere berekeningen op deze order verkeerde artikelprijzen worden gebruikt.

Voor de totaalbedragen ligt dit anders: deze blijven de som van de orderregelbedragen en zijn dus blijvend afleidbaar. Het is daarom niet strikt noodzakelijk ze in de database op te nemen. Opname heeft echter het voordeel dat de waarde direct beschikbaar is en niet steeds hoeft te worden berekend wanneer die wordt opgevraagd.

Uitwerkingen hoofdstuk 2

2.1a Gerecht en Eenheid
b Ingredient
c Ingredient

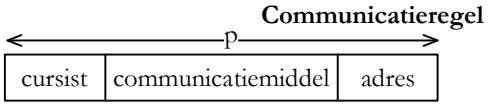
2.2 Omdat Boek.rubriek optioneel is, hoort bij een boek ofwel geen ofwel precies één rubriek. Dus: 0 of 1. Diagram c valt daardoor af. Omdat bij het strokendiagram geen extra regel gegeven is, is het toegestaan rubrieken op te nemen zonder dat in de database ook een boek van die rubriek is opgenomen. Dus: bij een rubriek horen 0 of meer (willekeurig veel) boeken. Ook diagram a valt daardoor af, zodat alleen diagram b in overeenstemming is met het multipliciteitendiagram.

2.3 Boek.rubriek is nu verplicht, zodat bij elk boek precies één rubriek hoort. Alleen diagram c klopt met het strokendiagram.

2.4 De kwalificaties 'ouder' en 'kind' gelden altijd *relatief* ten opzichte van een verwijzing. In figuur 2.13 zijn Filiaal en Werknemer ouder respectievelijk kind ten opzichte van de verwijzing 'is werkzaam bij'. Ten opzichte van de verwijzing 'heeft als regiomanager' is juist Filiaal het kind en Werknemer de ouder.
Onze tekenconventie om de 'ouder' altijd boven het 'kind' te tekenen, kunnen we in dit soort situaties natuurlijk niet volhouden.
Opmerking: een realistische database voor het grootwinkelbedrijf zou ten minste ook een tabel Regio bevatten, waardoor deze structuur niet precies zo zou voorkomen.

2.5a Foutmelding vanwege (een dreigende overtreding van) de referentiële-integriteitsregel: vanille komt niet voor in Product.naam.
b Foutmelding vanwege de primaire sleutel van Ingredient: de combinatie ('Coupe Kiwano', 'suiker') bestaat al.
c Foutmelding vanwege de referentiële-integriteitsregel: 'koffielepel' bestaat niet in Eenheid.naam.
d De rij wordt toegevoegd (het rdbms 'weet' niet, zoals wij, dat slagroom geen eenheid is).
e Foutmelding vanwege de referentiële-integriteitsregel en de restricted delete bij de verwijzing van Ingredient naar Product.
f De rij wordt verwijderd.
g De rij wordt verwijderd, samen met alle bijbehorende Ingredient-rijen, omdat voor de verwijzing van Ingredient naar Gerecht een cascading delete geldt.
h De verandering wordt doorgevoerd, niet alleen in Product, maar ook in alle voorkomens van 'kiwano' in de kolom Ingredient.product, als gevolg van de cascading update die geldt voor de verwijzing van Ingredient naar Product.
i Foutmelding vanwege de referentiële-integriteitsregel: 'pond' is geen bestaande eenheid.
j Foutmelding omdat Gerecht.bereidingstijd verplicht is.

2.6 Er is één brede uniciteitsregel(zie figuur 2.41).



Figuur 2.41

Deze houdt in dat smallere regels *niet* gelden. Er is namelijk geen reden voor bepaalde beperkingen, zoals de volgende voorbeeldpopulatie illustreert (zie figuur 2.42).

Communicatieregel

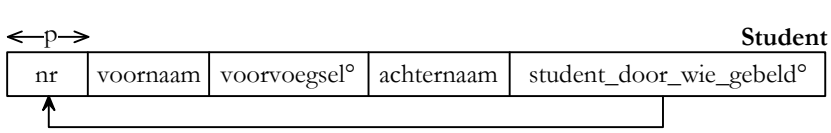
cursist	communicatiemiddel	adres
1	telefoon	020-1234567
1	telefoon	06-13579864
1	fax	020-1234567
2	telefoon	020-1234567

Figuur 2.42

Toelichting: de volgende beweringen geven aan waarom de drie mogelijke uniciteitsregels over twee kolommen niet gelden:
– een student kan twee verschillende telefoonnummers hebben
– een student kan een telefoonnummer delen met een andere student
– een student kan één nummer hebben voor zowel telefoon als fax.

2.7 De wie-belt-wie-associatie is 'één-op-veel' en kan worden gerealiseerd met een recursieve verwijzing van Student naar zichzelf. Zie het strokendiagram van figuur 2.43a en de voorbeeldpopulatie van figuur 2.43b.

a



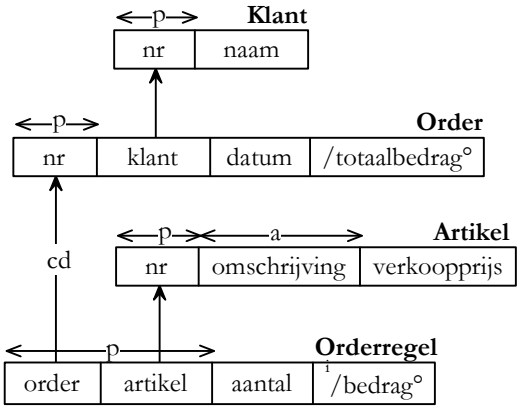
b

Student

nr	voornaam	voorvoegsel	achternaam	student_door_wie_gebeld
1	Inge		Berk	
2	Max		Tack	1
3	Wim		Bos	2
4	Iris	van	Eik	2
5	Sofie	van	Esch	1
6	Pim		Hazelaar	5
7	Cees	in 't	Woud	5
8	Anna		Roos	6

Figuur 2.43

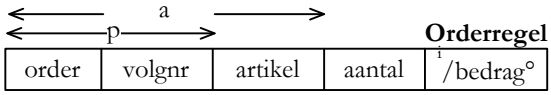
2.8 Zie figuur 2.44.



Figuur 2.44

Voor de verwijzing van Orderregel naar Order hebben we een cascading delete gespecificeerd: omdat een order één is met zijn orderregels impliceert het (willen) verwijderen van een order het (willen) verwijderen van al zijn orderregels.

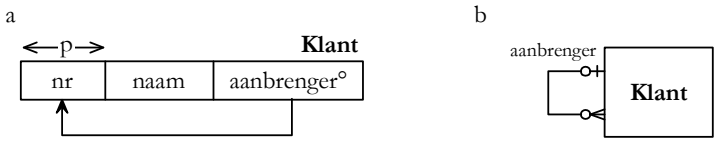
2.9 Orderregel krijgt twee, overlappende uniciteitsregels (zie figuur 2.45).



Figuur 2.45

Nu kunnen we kiezen welke we als primaire sleutel nemen, de ander wordt dan vanzelf alternatieve sleutel. De combinatie (order, volgnr) is onze favoriet voor de primaire sleutel. De reden is dat het bestaansrecht van een volgnummer helemaal is gelegen in de unieke identificatie van een orderregel binnen een order. De uniciteitsregel over (order, volgnr) zal daarom ook nooit vervallen. Daarentegen is best denkbaar dat geen uniciteitsregel geldt over (order, artikelnr): we zouden immers kunnen toelaten dat een order meerdere orderregels bevat over hetzelfde artikel.

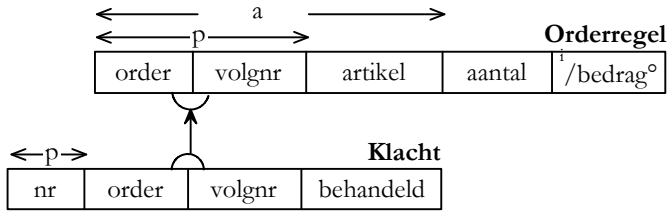
2.10 Om het aanbrengen van klanten door klanten te kunnen administreren is een recursieve verwijzing nodig van Klant naar Klant. Figuur 2.46 geeft een strokendiagram (figuur a) en een multipliciteitendiagram (figuur b).



Figuur 2.46

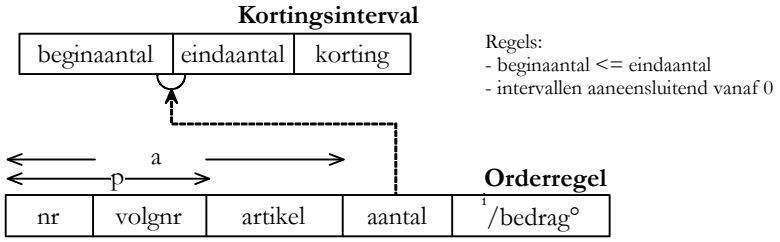
Als constraint zou nog toegevoegd kunnen worden: 'een klant mag niet zichzelf aanbrengen'.

2.11 Zie figuur 2.47.



Figuur 2.47

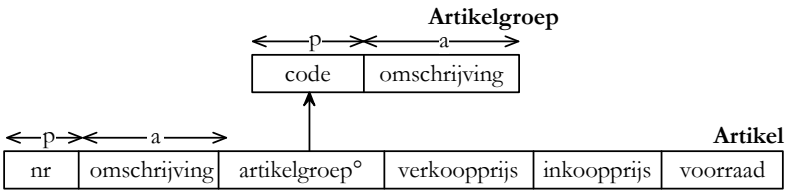
2.12 Een ontwerpvraag is hier: als kortingspercentages veranderen, moeten dan de oude kortingspercentages (gebruikt voor bestaande orderregels) nog bekend blijven? Zo ja, dan is het gewenst Orderregel een extra kolom korting te geven, die 'on insert' automatisch wordt gevuld en daarna onveranderbaar is. We zullen van deze mogelijkheid afzien, het strokendiagram wordt dan als volgt uitgebreid (zie figuur 2.48).



Figuur 2.48

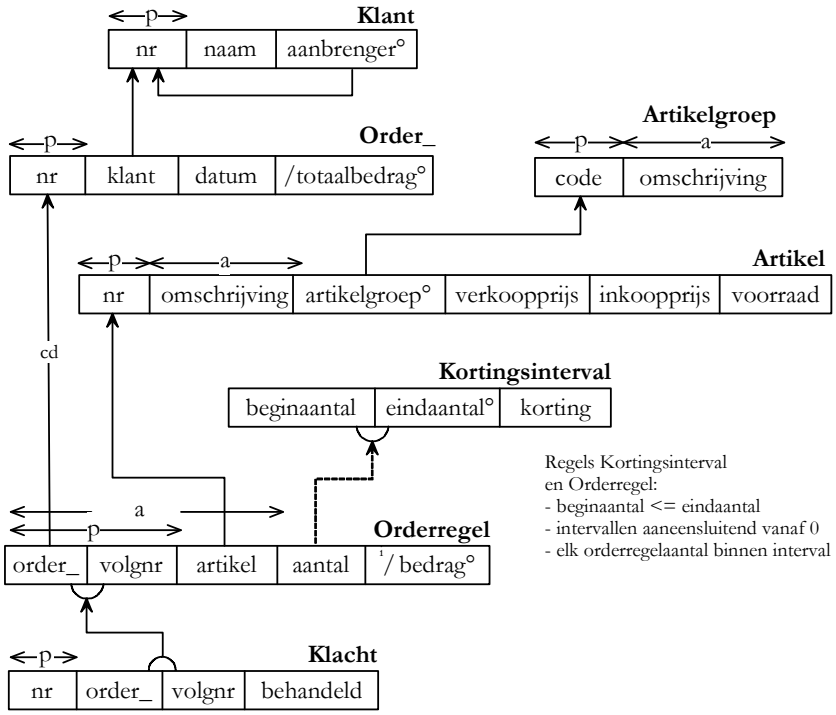
Opmerkingen
– Merk op dat voor Kortingsinterval geen sleutels zijn gedefinieerd. In plaats daarvan is een regel opgenomen die garandeert dat de intervalverwijzingen voor alle niet-negatieve aantallen (tot aan een zeker maximum) correct werken. Als die (zeer sterke) regel wordt afgedwongen – hoe, dat zien we later wel – heeft het geen zin om Kortingsinterval ook nog te controleren op een primaire sleutel voor beginaantal of een alternatieve sleutel voor eindaantal. Primaire en alternatieve sleutels zijn hier als regel te zwak. Alleen de sterkste regels hoeven te worden bewaakt.
– Als gevolg van deze wijziging moet de regel voor de automatische berekening van de orderregelbedragen worden aangepast.

2.13 Zie figuur 2.49.



Figuur 2.49 Fragment strokendiagram met Artikel en Artikelgroep

2.14 Zie figuur 2.50.



Figuur 2.50 Volledig strokendiagram Orderdatabase

Zie de bijlage achterin het boek voor een voorbeeldpopulatie.

Uitwerkingen hoofdstuk 3

3.1-3.5 Geen uitwerking.

3.6 SQL-statement:

```
select   product, hoeveelheidPP
from     Ingredient
where    gerecht = 'Coupe Kiwano'
order by volgnr
```

3.7 SQL-statement:

```
select naam, bereidingstijd
from   Gerecht
where  bereidingstijd >= 5 and bereidingstijd <= 10
```

Het kan ook met de (in leereenheid 7 te behandelen) `between ... and`-operator:

```
select naam, bereidingstijd
from   Gerecht
where  bereidingstijd between 5 and 10
```

3.8a De volgorde in `delete`-statements is belangrijk: van kind naar ouder. Een poging om een oudertabel leeg te maken, zal immers mislukken, als de kindtabel nog één of meer rijen bevat met een verwijzing.

b Kolom energiePP wordt in eerste instantie gevuld met nulls: `insert into Gerecht values (..., null, ..., ...)`. Verderop in het script echter, bij het toevoegen van de ingrediëntrijen, vindt na elke toevoeging een automatische berekening of herberekening plaats van energiePP. Dit gebeurt door de werking van 'triggers' (programmaatjes in de database).

3.9 De toevoegingen kunnen in een script worden gebundeld:

```
insert into Gerecht values ('Reijnders Roem', null, 12,
    'Snijd het fruit in stukjes, vermeng alles met de pernod, '
    || 'en laat dit een uurtje staan alvorens het op te dienen. ');
insert into Product values ('meloen', 'stuks', 150);
insert into Ingredient values ('Reijnders Roem', 'kiwano', 0.5, null);
insert into Ingredient values ('Reijnders Roem', 'mango', 0.2, null);
insert into Ingredient values ('Reijnders Roem', 'meloen', 0.1, null);
insert into Ingredient values ('Reijnders Roem', 'pernod', 1, null)
```

Opmerkingen

- De volledige string voor de bereidingswijze van Reijnders Roem is te lang voor één regel. Omdat binnen een string geen overgang naar een nieuwe regel is toegestaan, wordt de string opgedeeld in deelstrings, die tot één string worden geconcateneerd.
- De energie per persoon (ingevoerd als `null`) wordt automatisch berekend door een trigger. Dit geldt ook voor de ingrediënt-volgnummers. De nulls mogen ook worden weggelaten, mits u de lijst van de overige kolommen in het `insert`-statement opneemt, bijvoorbeeld:

```
insert into Ingredient (gerecht, product, hoeveelheidPP)
values ('Reijnders Roem', 'kiwano', 0.5)
```

3.10 Foutmelding: voor verplicht veld is `null` niet toegestaan.

3.11 Foutmelding: overtreding van de referentiële-integriteitsregel. Eerst moet de eenheid worden toegevoegd, pas daarna kunt u een product toevoegen met een verwijzing naar de nieuwe eenheid.

3.12 SQL-statement:

```
update Gerecht
set    bereidingstijd = 30
where  naam = 'Coupe Kiwano'
```

3.13 Het statement:

```
delete
from  Product
where naam = 'ijs'
```

geeft een foutmelding: deze Product-rij heeft nog 'kinderen' waarnaar vanuit Ingredient wordt verwezen.

3.14 In het algemeen geldt: om niet in strijd te komen met de referentiële-integriteitsregel dienen de rijen 'van beneden naar boven' verwijderd te worden, ofwel: van kind naar ouder. Vanwege de cascading delete voor de verwijzing van Ingredient naar Gerecht worden echter met een gerecht automatisch ook de bijbehorende ingrediëntregels verwijderd:

```
delete
from  Gerecht
where naam = 'Reijnders Roem';
```

Nu is de weg vrij om ook het product 'meloen' te verwijderen:

```
delete
from  Product
where naam = 'meloen';
```

Ongedaan maken van alle veranderingen sinds het vorige commitmoment:

```
rollback
```

3.15 Nee, het valt 'gewoon' te begrijpen: omdat een `delete` op rijniveau werkt (u kunt alleen maar hele rijen verwijderen), heeft het geen zin kolommen te vermelden, zelfs niet een * voor 'alle kolommen'.

3.16 Als extra experiment zou u de Toetjesboek-database kunnen de-installeren door vanuit IQU de opdracht `drop database` te geven. Er is nog een derde mogelijkheid: de-installatie door het bestand Toetjesboek.fdb via Windows te verwijderen. Het effect blijkt in alle gevallen volstrekt identiek te zijn!

3.17 Geef een toepasselijk `delete`-statement voor Gerecht (de oudertabel) en constateer dat hierna de bijbehorende rijen in Ingredient (de kindtabel) eveneens zijn verwijderd. Eindig met `rollback`.

3.18 Geef een toepasselijk `delete`-statement voor Product (de oudertabel) en bestudeer de foutmelding.

3.19 Wijzig een primaire-sleutelwaarde in Product (bijvoorbeeld 'zure room' in 'sour cream'), en constateer dat deze waarde in Ingredient.productnaam eveneens is gewijzigd.

3.20 SQL-statement:

```
select   nr, naam
from     Klant
order by naam
```

3.21 SQL-statement:

```
select *
from   Orderregel
where  artikel = 107 or artikel = 180
```

3.22a Toevoeging:

```
insert into Artikel values (470, 'zwamdop', null, 17, 12.50, 1000)
```

De `null` in de waardenlijst mag worden weggelaten; dan is echter een kolomlijst voor de andere waarden verplicht:

```
insert into Artikel(nr, omschrijving, verkoopprijs, inkoopprijs, voorraad) values (470, 'zwamdop', 17, 12.50, 1000)
```

b Wijziging:

```
update Artikel
set    omschrijving = 'zwemdop',
       verkoopprijs = 15
where  omschrijving = 'zwamdop'
```

c Verwijdering:

```
delete
from  Artikel
where omschrijving = 'zwemdop'
```

3.23 Een poging tot invoegen van een artikel met een al bestaand artikelnummer, bijvoorbeeld:

```
insert into Artikel values (351, 'borgstekker', 'VR', 3.95, 2.80, 910)
```

leidt tot de foutmelding: violation of primary key or unique key constraint "INTEG_19" on table "ARTIKEL".

Opmerking: de constraintnaam INTEG_19 is door het rdbms automatisch gegenereerd. Afhankelijk van de voorgeschiedenis zou het nummer kunnen afwijken.

3.24 Een poging tot invoegen van een order met een klantnummer-zonder-klant, bijvoorbeeld:

```
insert into Order_values (5803, 1250, '20-jan-2002', null)
```

leidt tot de foutmelding: violation of foreign key constraint "INTEG_9" on table "ORDER_". (Het nummer in de constraintnaam kan ook hier afwijken.)

3.25 SQL-statement:

```
update Klant
set    nr = nr + 1000
```

Via een select-opdracht op Order_ (projectie op klant) kunnen we nu vaststellen dat de klantnummers in Order_ eveneens met 1000 zijn verhoogd.

3.26 Er wordt, voordat wordt gecontroleerd op de restricted delete bij de verwijzing van tabel Order_ naar tabel Klant, eerst gecontroleerd op een andere regel: de recursieve verwijzing van tabel Klant naar zichzelf. Dit blijkt bij de deletepoging van bijvoorbeeld klant Roos, die zelf klantaanbrenger is (violation of foreign constraint ... on table Klant), versus de deletepoging van klant Voskuil, die geen aanbrenger is (violation of foreign constraint ... on table Order_).

Wanneer geen delete-regel is gespecificeerd, geldt de referentiële-integriteitsregel. Dit komt neer op een restricted delete.

3.27 Het eerste statement leidt in eerste instantie tot een *poging* om de Order_-rij met nr 5773 te verwijderen. Omdat voor de verwijzing van Orderregel naar Order_ een cascading delete geldt, leidt dit tot een poging om de Orderregel-rijen met order 5773 te verwijderen. Dit lukt, omdat er geen Klacht-rijen zijn die naar een van die rijen verwijzen.

Het tweede statement leidt tot analoge pogingen, onder meer om de Orderregel-rij met order 5774 en volgnr 1 te verwijderen. Dit lukt echter niet, omdat er een Klacht-rij bestaat die hiernaar verwijst. Het resultaat is dat order 5774 met al zijn orderregels blijft bestaan.

3.28 De energiewaarden per persoon in kilocalorie worden als databasewaarde bewaard (kolom energiePP) en automatisch berekend zodra de betreffende Gerecht-rij (met de corresponderende rijen in de andere tabellen) is ingevoerd. Automatische herberekening vindt plaats zodra één van de gegevens die op de berekening van invloed zijn, is gewijzigd. De energiewaarden per persoon in kilojoule worden niet in de database bewaard. Ze komen alleen beschikbaar in de applicatie en hun berekening vindt pas plaats op het moment dat ze in een Gerecht-venster moeten worden getoond.

3.29 en **3.30** Geen uitwerking.

Uitwerkingen hoofdstuk 4

4.1 Het resultaat is: 'no records returned'. De verklaring is dat elke expressie
waarde = `null` als logische uitkomst een *unknown* heeft. Dit is zelfs het geval wanneer *waarde* zelf `null` is.

4.2 We zijn nu gedwongen voor 'lege' toonhoogten een defaultwaarde in te vullen. De spatie lijkt hiervoor het meest aangewezen, omdat we die 'toch niet zien'. Dit blijkt echter niet helemaal waar te zijn, zoals blijkt uit het statement:

```
select toonhoogte || naam
from   Instrument
```

met het volgende wat slordige resultaat:

```
===============
 piano
 fluit
altfluit
altsaxofoon
tenorsaxofoon
sopraansaxofoon
 gitaar
 viool
altviool
 drums
```

De beginspaties kunnen er met een geschikte functie wel 'afgepeld' worden, maar mooi is anders. Een andere kandidaat-defaultwaarde is de lege string. Deze is echt onzichtbaar. Of het goed gaat, is echter dialectafhankelijk. In sommige dialecten heeft de lege string de waarde null, in andere is het een echte string, met lengte 0. Als de lege string de waarde null heeft, is het resultaat afhankelijk van hoe een null zich gedraagt bij concatenatie.

4.3 SQL-statement:

```
select *
from   Inschrijving
where  ((cursus = 'DB' or cursus = 'IM') and vrijstelling is not null)
       or cursus = 'II'
```

Opmerkingen
- Het buitenste haakjespaar staat er alleen voor de duidelijkheid.
- Merk op dat 'en' in natuurlijke taal aanleiding geeft tot `or` in de code. Het omgekeerde komt ook voor: dat 'of' in natuurlijke taal leidt tot `and`.
- Een alternatief voor de expressie `cursus = 'DB' or cursus = 'IM'` is de expressie: `cursus in ('DB', 'IM')`. De operator `in` is de verzamelingenoperator 'is element van'.

4.4 Statement a geeft alle inschrijvingen, statement b alleen die waarvoor een cijfer is ingevuld. De selectieconditie in statement c geeft *unknown* `or` *unknown* voor de inschrijvingen zonder cijfer, wat *unknown* oplevert. Aangezien '`where` *unknown*' hetzelfde effect heeft als '`where` *false*' bevat het resultaat alleen inschrijvingen met een cijfer, net als statement b. Tenslotte evalueert in statement d de conditie 1 = 1 tot *true*, ongeacht de actuele rij. Voor elke rij staat er dus eigenlijk '`where` *true*'. Dus geeft ook deze query alle inschrijvingen, net als statement a.

4.5 De logisch equivalente groepjes zijn: (a, f); (b, d, g); c; e.

4.6 De logische expressies a en d zijn tautologisch (altijd *true*) en dus equivalent. De expressies b en c zijn eveneens equivalent, omdat de operator <> per definitie de negatie is van de =-operator. Tot slot is expressie e equivalent met c (en dus ook met b), vanwege onder meer de tweede regel van De Morgan.
De expressies a en d enerzijds zijn echter niet equivalent met b, c en e anderzijds, omdat b, c en e ook de logische waarde *unknown* kunnen aannemen (namelijk als de artikelgroep null is).

4.7 Alle artikelen met code 'VR' (voegringen) die niet in voorraad zijn:

```
select nr, omschrijving
from   Artikel
where  artikelgroep = 'VR' and voorraad = 0
```

Voor het overzicht van de overige artikelen ligt het voor de hand de `where`-clausule te voorzien van een negatie:

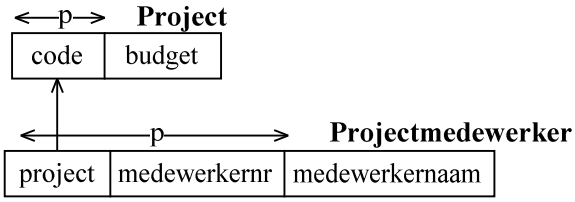
```
select nr, omschrijving
from   Artikel
where  not(artikelgroep = 'VR' and voorraad = 0)
```

Dit is echter niet correct. We hebben immers te maken met uitdrukkingen in een driewaardige logica en de logische waarden van <selectieconditie> en van `not` <selectieconditie> zijn in dit geval niet complementair. We missen dan ook de artikelen waarvoor de selectieconditie de waarde *unknown* heeft: de losse artikelen. Een correcte oplossing is:

```
select nr, omschrijving
from   Artikel
where  not(artikelgroep = 'VR' and voorraad = 0)
       or artikelgroep is null
```

Uitwerkingen hoofdstuk 5

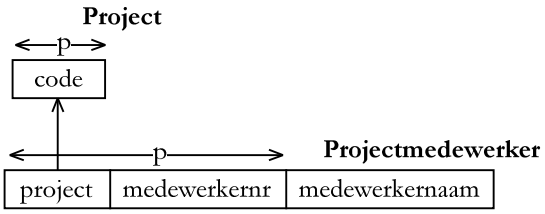
5.1a Bij afsplitsing van een herhalende groep wordt de primaire sleutel 'meegenomen' in de vorm van een verwijssleutel. Het resultaat is een 'kind' met een verwijzing naar een 'ouder'. Vaak (niet altijd) wordt in de kindtabel de verwijssleutel onderdeel van een brede primaire sleutel. Hier is dat inderdaad het geval (zie figuur 5.32).



Figuur 5.32

De tabelnaam Projectmedewerker is gekozen omdat de tabel projecten koppelt aan medewerkers. De kolomnamen nr en naam zijn aangepast, om ook in nieuwe context duidelijk te maken dat het kenmerken zijn van een medewerker.

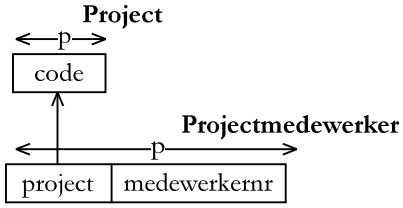
b Er verandert niets wezenlijks (zie figuur 5.33).



Figuur 5.33

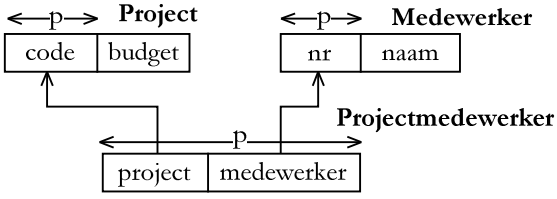
Vanuit onderdeel a gezien is dit vrij logisch, toch blijkt dit randgeval vaak lastig gevonden te worden.

c Dit onderdeel illustreert een herhalende groep met één kolom. Dit is nog meer een randgeval dan onderdeel b. Maar ook nu verandert er niets wezenlijks (zie figuur 5.34).



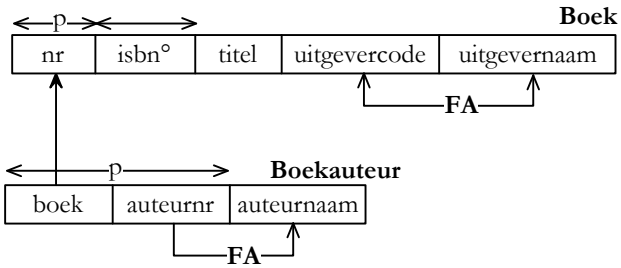
Figuur 5.34

5.2 In de figuren 5.32 (onderdeel a) en 5.33 (onderdeel b) bevat tabel Projectmedewerker een partiële sleutelafhankelijkheid: nr → naam. De structuren voldoen dus niet aan 2NV. In figuur 5.35 tonen we de genormaliseerde structuur voor onderdeel a. (Voor onderdeel b gaat het analoog.)



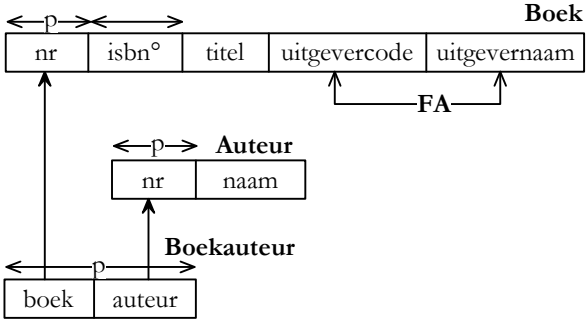
Figuur 5.35

5.3 Na afsplitsen van de herhalende groep staat de structuur in 1NV (zie figuur 5.36).



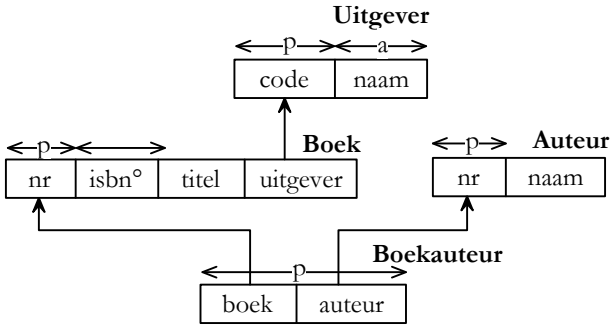
Figuur 5.36

De afgesplitste tabel heeft de naam Boekauteur gekregen. Elke rij bevat immers auteurinformatie in relatie tot een bepaald boek. Auteurregel was ook een goede naam geweest.
Merk op dat uitgevercode en uitgevernaam wederzijds functioneel afhankelijk zijn. In plaats van met twee aparte FA-symbolen is dit weergegeven met één FA-symbool met twee pijlpunten.
De functionele afhankelijkheid binnen Boekauteur impliceert een partiële sleutelafhankelijkheid. De structuur staat daardoor niet in 2NV. Na decompositie (afsplitsing van tabel Auteur) is dit wel het geval (zie figuur 5.37).



Figuur 5.37

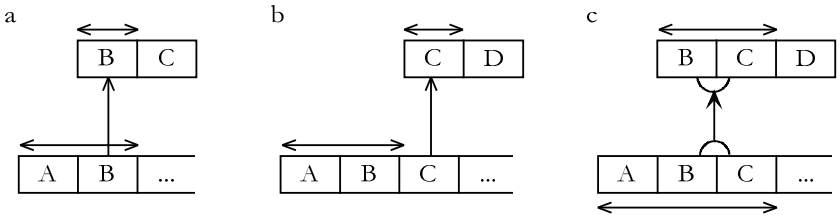
De twee getekende functionele afhankelijkheden binnen tabel Boek zijn van het type 'transitieve sleutelafhankelijkheid', zodat Boek, en daarmee de hele structuur, niet in 3NV staat.
Er zijn nu twee manieren van afsplitsen mogelijk, afhankelijk van wat we als primaire sleutel willen in de afgesplitste tabel: uitgevercode of uitgevernaam. Omdat bij de keuze van een primaire sleutel korte codes de voorkeur verdienen boven een langere naam, valt de keuze op uitgevercode (zie figuur 5.38).



Figuur 5.38

Merk op dat we de kolomnamen 'auteurnaam' en 'uitgevernaam' hebben veranderd in 'naam'. In de context van de tabellen Auteur en Uitgever zijn nadere kwalificaties immers niet nodig.

5.4 Zie figuur 5.39.



Figuur 5.39

5.5 Bij een gegeven waarde a voor kolom A hoort precies één waarde b voor kolom B. Zou nu een waarde a twee keer voorkomen, dan zou de combinatie (a, b) ook twee keer voorkomen, in strijd met de gegeven uniciteitsregel over A en B. Dus geldt een uniciteitsregel over kolom A. Dit is echter in strijd met de conventie om alleen de meest strenge uniciteitsregels als uniciteitspijl te noteren.

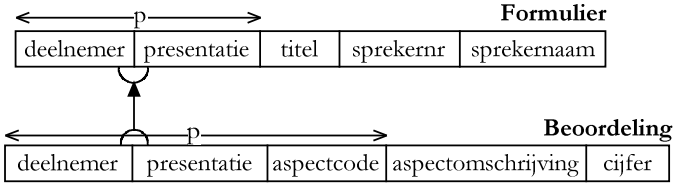
5.6 We lopen de mogelijkheden na voor een FA: B → C met niet-unieke determinant. Mocht C een kolomcombinatie zijn, dan determineert B ook elke afzonderlijke kolom. Zonder verlies van algemeenheid nemen we daarom nu aan dat C één kolom is.

a C is géén sleutelkolom. Alle mogelijkheden voor B worden in dit geval afgedekt door de figuren 6.9 (overtreding 2NV) en 6.13 (overtreding 3NV).

b C is wel een sleutelkolom. Als C een volledige sleutel is, moet B dat ook zijn, in tegenspraak met de aanname dat de FA een niet-unieke determinant heeft. Blijft dus over dat C een deel is van een sleutel: precies de situatie van figuur 5.27.

5.7 Indien, naast de uniciteitsregel, geen enkele beperkingsregel is gegeven, voldoet de structuur aan BCNV. (Dit was een strikvraag, om nog eens onder de aandacht te brengen dat uit een populatie nooit regels zijn af te leiden; hooguit kan worden afgeleid dat een bepaalde regel *niet* geldt.)

5.8a Afsplitsen van de herhalende groep geeft de structuur in 1NV van figuur 5.40. Hierbij is de naamgeving verbeterd: de nieuwe kolomnamen aspectcode en aspectomschrijving geven expliciet aan om wat voor code en omschrijving het gaat. Het zijn immers geen codes en omschrijvingen van beoordelingen maar van aspecten waarop beoordeeld wordt. D.m.v. een 'p' is aangegeven dat bij de uniciteitsregels een primaire sleutel hoort.

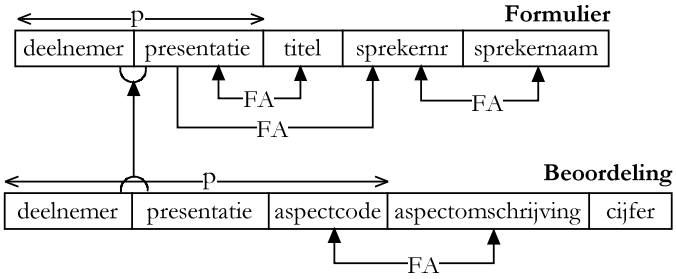


Figuur 5.40

b De structuur van figuur 5.40 bevat een aantal functionele afhankelijkheden, die tot redundantie aanleiding kunnen geven. Er zijn drie paren van kolommen die wederzijds functioneel afhankelijk zijn:
- bij een presentatie(nummer) hoort één titel en omgekeerd
- bij een sprekernr hoort één sprekernaam en omgekeerd
- bij een aspectcode hoort één aspectomschrijving en omgekeerd

Daarnaast geldt in tabel Formulier:
- bij een presesentatie(nummer) hoort één sprekernr.

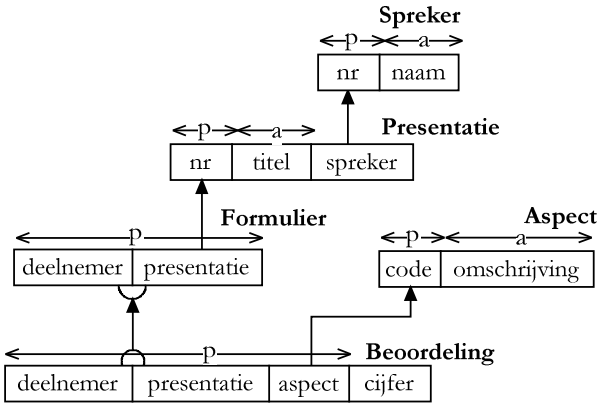
Zie figuur 5.41.



Figuur 5.41

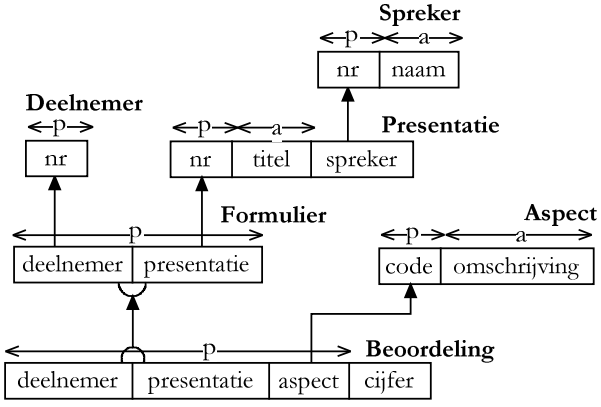
Opmerkingen
- Omdat in tabel Formulier presentatie en titel wederzijds functioneel afhankelijk zijn, zijn deze beide kolommen gelijkwaardig als het om identificatie van een rij gaat. Daarom is er naast de primaire sleutel (deelnemer, presentatie) nog een alternatieve sleutel (deelnemer, titel). Deze hebben we niet getekend.
- Om analoge redenen geldt: in tabel Beoordeling is (deelnemer, presentatie, aspectomschrijving) een alternatieve sleutel.

c Afsplitsen van de FA's geeft een resultaat dat voldoet aan 3NV (en ook BCNV), zie figuur 5.42.



Figuur 5.42

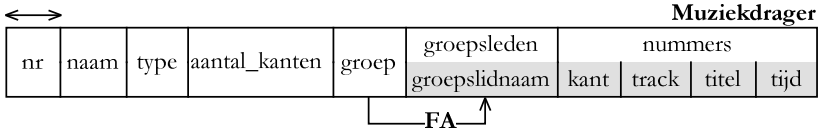
d De 'wereld' van de conferentie bevat een entiteittype dat nog geen eigen tabel heeft: het entiteittype 'deelnemer'. De deelnemernummers zijn hierdoor niet gestandaardiseerd. Ze worden 'gewoon' gebruikt in tabel Formulier. Al is het deelnemernummer het enige hier gebruikte kenmerk van deelnemers, dit is geen reden om zo'n eigen tabel niet te maken. Figuur 5.43 bevat de verbeterde structuur. Uiteraard kunnen andere kenmerken aan de tabel Deelnemer worden toegevoegd, zoals naam, e-mail, enzovoort.



Figuur 5.43

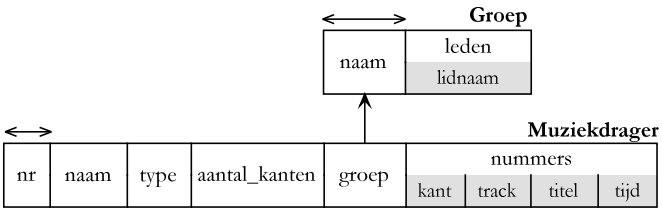
Let wel: deze stap maakt geen deel uit van het normalisatieproces. Mede hierom is normalisatie geen volwaardige modelleermethode.

5.9a Figuur 4.44 toont de gewraakte functionele afhankelijkheid.



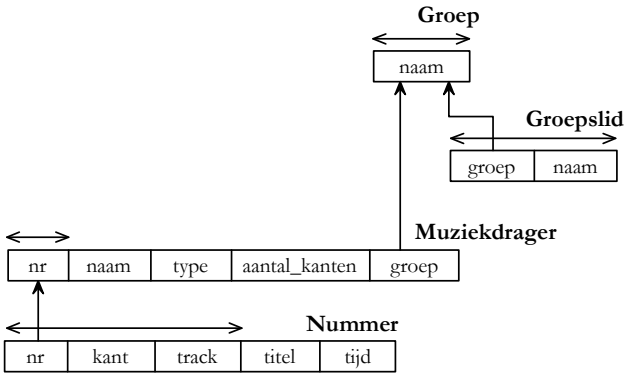
Figuur 5.44

Afsplitsen resulteert in de structuur van figuur 5.45.



Figuur 5.45

b Normalisatie tot 1NV leidt tot figuur 5.46. Zowel Groep als Muziekdrager krijgt er een kindtabel bij voor de meervoudige kenmerken uit hun respectievelijke herhalende groepen.



Figuur 5.46

Merk op dat groepsnamen tengevolge van deze operatie zijn gestandaardiseerd in de 'eigen' tabel Groep, waarbij de groepsnamen in de tabellen Groepslid en Muziekdrager zijn onderworpen aan de referentiële integriteitsregel.

c en d Dergelijke functionele afhankelijkheden zijn er niet. De structuur van onderdeel b staat dus al in 3NV (en ook in BCNV).

Uitwerkingen hoofdstuk 6

6.1 SQL-statement:

```
select distinct examinator
from   Cursus
where  examinator is not null
```

6.2 SQL-statement:

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

6.3 SQL-statement:

```
select nr, naam, coalesce(mentor, 'geen mentor') mentor
from   Student
```

6.4 De volgende `select`-expressie:

```
select to_date ('15-okt-1582') - to_date ('4-okt-1582')
from   Rdb$database
```

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

6.5 Oplossing met `or`:

```
select distinct cursus
from   Voorkenniseis
where  voorkennis = 'II' or voorkennis = 'DW' or voorkennis = 'IM'
```

Oplossing met `in`:

```
select distinct cursus
from   Voorkenniseis
where  voorkennis in ('II', 'DW', 'IM')
```

Resultaat:

```
CURSUS
======
DB
```

Merk op dat hier een `distinct` nodig is. Zonder `distinct` zouden cursussen die meer dan één cursus uit het rijtje als voorkennis eisen, meervoudig worden vermeld. (Bij de gegeven voorbeeldpopulatie zou 'DB' dubbel worden vermeld.) Met onze huidige kennis ligt een oplossing zonder de 'pseudo-operator' `distinct` (zie grijstekst aan einde van paragraaf 1) nog buiten ons bereik.

6.6a SQL-statement:

```
select *
from   Cursus
where  not(examinator = 'DAT') or examinator is null
```

b SQL-statement:

```
select *
from   Cursus
where  examinator is not null and not(examinator = 'DAT')
```

In beide query's kan `not(examinator = 'DAT')` vervangen worden door `examinator <> 'DAT'`. In de tweede query kan '`examinator is not null and`' achterwege blijven, omdat die eis impliciet ook al in de tweede operand zit opgesloten. Expliciete vermelding dat er een examinator moet zijn maakt de query echter veel duidelijker.

6.7 De volgende oplossing maakt gebruik van een alias:

```
select   naam, uren - 28 * credits verschil
from     Cursus
order by verschil desc, naam asc
```

Zonder alias kan het ook: `order by uren - 28 * credits desc, naam asc`. Ook kunt u volgnummers gebruiken: `order by 2 credits desc, 1 asc`

6.8a Oplossing met `union`:

```
select student, cursus, 'vrijstelling'
from   Inschrijving
where  vrijstelling = 'J'
union
select student, cursus, 'geen vrijstelling'
from   Inschrijving
where  vrijstelling = 'N'
union
select student, cursus, 'vrijstelling onbekend'
from   Inschrijving
where  vrijstelling is null
order by 3, 1, 2
```

b Oplossing met case:

```
select student, cursus,
       case
          when vrijstelling = 'J' then 'vrijstelling'
          when vrijstelling = 'N' then 'geen vrijstelling'
          when vrijstelling is null then 'vrijstelling onbekend'
       end
from   Inschrijving
order  by 3, 1, 2
```

6.9 SQL-statement:

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

6.10 SQL-statement:

```
select 'Artikel ' || nr || ' kost '
       || cast(round(verkoopprijs * 1.19, 2) as numeric(6,2))
       || ' euro '  prijslijst
from   Artikel
```

6.11 SQL-statement:

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

6.12 Artikelen met en zonder artikelgroep worden apart behandeld en samengevoegd 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
```

Opmerking: in de laatste regel is `order by nr` niet toegestaan.

6.13 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.

Uitwerkingen hoofdstuk 7

7.1a Vanwege de cursusnaam moeten we Inschrijving joinen met Cursus. Vervolgens vindt selectie plaats op de aanvullende voorwaarden:

```
select I.student, C.naam, I.datum, I.cijfer, I.vrijstelling
from   Inschrijving I, Cursus C
where  I.cursus = C.code
       and (I.cijfer is null or I.cijfer <= 5)
       and (I.vrijstelling = 'N' or I.vrijstelling is null)
```

b De combinatie (Inschrijving.student, Inschrijving.cursus) is, als primaire sleutel, identificerend. Hiervan is Inschrijving.cursus (met een cursuscode) voor identificatie equivalent is met de cursusnaam. Immers Cursus.naam is een alternatieve sleutel.

7.2 De laatste coderegel 'and C.examinator is not null' is overbodig, omdat cursussen zonder examinator toch al niet in het resultaat voorkomen. Immers, als C.examinator null is, herleidt de eerste `conditie` tot: null = D.acr en dat geeft een logische *unknown*. Daarmee wordt de hele `where`-conditie *unknown*. En '`where` *unknown*' geeft hetzelfde resultaat als '`where` *false*'.

7.3a De lege resultaatrij voor peper wordt veroorzaakt doordat I.hoeveelheidPP en P.eenheid dan null zijn en null geeft in Firebird bij concatenatie null als resultaat.

b Voor 'gewone' ingrediënten doet de coalesce-aanroep niets en is het statement equivalent met de vereenvoudigde vorm. Voor bijvoorbeeld peper is het eerste argument van coalesce null en wordt dus het tweede argument geretourneerd: de lege string. Indien deze wordt geconcateneerd met de productnaam, is het resultaat die productnaam, precies wat we voor 'peper' nodig hebben.

7.4 We bouwen de oplossing op uit twee union-operands. De eerste geeft de cursussen met examinator, de tweede geeft de cursussen zonder examinator:

```
select C.naam, C.examinator, D.naam, D.vervanger
from   Cursus C, Docent D
where  C.examinator = D.acr
union
select naam, examinator, null, null
from   Cursus
where  examinator is null
```

7.5 Er is geen verschil, omdat de verwijssleutel (over kolom Inschrijving.cursus) verplicht is.

7.6 SQL-statement:

```
select C.code, C.naam, C.credits, C.examinator, D.naam
from   Cursus C
       left outer join Docent D on C.examinator = D.acr
where  C.credits >= 4
```

7.7 SQL-statement:

```
select C.code, C.naam,
       coalesce(C.examinator, 'geen examinator') examinator,
       D.naam,
       coalesce(D.vervanger, 'geen vervanger') vervanger
from   Cursus C
       left outer join Docent D on C.examinator = D.acr
```

7.8 De oplossing is een left outer join, met de oudertabel voorop:

```
select C.code, C.naam, B.docent begeleider
from   Cursus C
       left outer join Begeleider B on C.code = B.cursus
```

Resultaat:

```
CODE   NAAM                 BEGELEIDER
====== ==================== =========
II     Inleiding informatica BAC
DW     Discrete wiskunde     DAT
DB     Databases             DAT
DB     Databases             BAC
IM     Informatiemodelleren  DAT
SW     Semantic web          <NULL>
```

Dit is de genormaliseerde vorm (1NV) van het overzicht van figuur 7.17. De derde kolom daarin vormt een herhalende groep (vijf subtabellen van één kolom en één of meer rijen). Zo'n overzicht kan niet met SQL-verkregen worden. Hiervoor is een rapportgenerator nodig.

```
CODE   NAAM                 BEGELEIDERS
                            (ACR)
====== ==================== ==========
II     Inleiding informatica BAC
DW     Discrete wiskunde     DAT
DB     Databases             DAT
                             BAC
IM     Informatiemodelleren  DAT
SW     Semantic web          <NULL>
```

Figuur 7.17 Cursustabel met herhalende groep van begeleiderinformatie

7.9 SQL-statement:

```
select T.student, T.cursus, T.datum tentamendatum,
       I.datum inschrijvingsdatum, T.cijfer
from   Tentamen T
       join Inschrijving I on T.student = I.student
                           and T.cursus = I.cursus
where  T.cijfer >= 6
```

7.10 De producttabel heeft $7 \times 10 \times 5 = 350$ rijen en $5 + 5 + 3 = 15$ kolommen.

7.11 De vraagstelling heeft betrekking op drie verschillende tabellen waarvan de join gevormd moet worden: tabel Voorkenniseis en twee exemplaren van de tabel Cursus, één in de rol van 'cursus waarvan we de voorkenniscursussen willen weten' en één in de rol van 'voorkenniscursus'. We geven deze exemplaren aliasnamen C respectievelijk Vc.

Oplossing met cartesisch-productoperator:

```
select C.naam, Vc.naam
from   Voorkenniseis Ve, Cursus C, Cursus Vc
where  Ve.cursus = C.code
       and Ve.voorkennis = Vc.code
```

Met joinoperator:

```
select C.naam, Vc.naam
from   Voorkenniseis Ve
       join Cursus C on Ve.cursus = C.code
       join Cursus Vc on Ve.voorkennis = Vc.code
```

7.12 De producttabel bevat $3 \times 3 = 9$ rijen en $3 + 3 = 6$ kolommen.

7.13 We moeten Student joinen met Docent over Student.mentor. Om de vervangers te krijgen moeten we doorjoinen met een tweede exemplaar van Docent over Docent.vervanger. Het eerste exemplaar van Docent geven we alias M (van 'mentor'), het tweede exemplaar alias VM ('vervanger van mentor'):

```
select S.nr, S.naam, M.naam mentor, VM.naam vervanger_mentor
from   Student S
       left outer join Docent M on S.mentor = M.acr
       left outer join Docent VM on M.vervanger = VM.acr
```

Resultaat:

```
      NR NAAM   MENTOR VERVANGER_MENTOR
======== ====== ====== ================
       1 Berk   C.Date E.Codd
       2 Tack   C.Date E.Codd
       3 Bos    E.Codd <NULL>
       4 Eik    <NULL> <NULL>
```

Merk op: als we van de eerste outer join een inner join maken, zouden we alleen maar de eerste twee rijen overhouden. Als we van de tweede outer join een inner join maken, zouden we de derde rij kwijtraken.

7.14 SQL-statement:

```
select D.naam, D.vakgroep, V.naam, V.vakgroep
from   Docent D
       join Docent V on D.vervanger = V.acr
where  D.vakgroep <> V.vakgroep
```

7.15 Met productoperator:

```
select O.nr, O.klant, O.datum, K.naam
from   Order_ O, Klant K
where  O.klant = K.nr
       and datum >= '01-nov-2011'
```

Met inner-joinoperator:

```
select O.nr, O.klant, O.datum, K.naam
from   Order_ O
       join Klant K on O.klant = K.nr
where  datum >= '01-nov-2011'
```

Opmerking: indien zeker is dat in datumwaarden de tijdcomponent 0 is, is de conditie gelijkwaardig met 'datum > 31-oct-2011'.

7.16 De eenvoudigste oplossing maakt gebruik van de operator `left outer join`:

```
select A.nr artikelnummer, A.omschrijving artikelomschrijving,
       A.artikelgroep groepcode, Ag.omschrijving groepomschrijving
from   Artikel A
       left outer join Artikelgroep Ag on A.artikelgroep = Ag.code
```

7.17 De `coalesce`-functie biedt uitkomst:

```
select A.nr artikelnummer, A.omschrijving artikelomschrijving,
       coalesce(A.artikelgroep, 'geen') groepcode,
       Ag.omschrijving groepomschrijving
from   Artikel A
       left outer join Artikelgroep Ag on A.artikelgroep = Ag.code
```

Een alternatief is dat we de outer join 'met de hand' maken, dat wil zeggen: vanuit de corresponderende inner join en een `union`-constructie:

```
select A.nr artikelnummer, A.omschrijving artikelomschrijving,
       A.artikelgroep groepcode, Ag.omschrijving groepomschrijving
from   Artikel A
       left outer join Artikelgroep Ag on A.artikelgroep = Ag.code
union
select nr, omschrijving, 'geen', null
from   Artikel
where  artikelgroep is null
```

7.18 SQL-statement:

```
select K.nr, Orl.artikel, A.omschrijving
from   Klacht K
       join Orderregel Orl on K.order_ = Orl.order_
                          and K.volgnr = Orl.volgnr
       join Artikel A on Orl.artikel = A.nr
```

7.19 De producttabel heeft 64 rijen en 6 kolommen (zie figuur 7.18).

Klant K, Klant A (Cartesisch product)

K.nr	K.naam	K.aanbrenger	A.nr	A.naam	A.aanbrenger
1234	Cupido		1234	Cupido	
1447	Roos	1234	1234	Cupido	
1448	Voskuil		1234	Cupido	
1449	Grunberg		1234	Cupido	
1450	Crosby		1234	Cupido	
1451	Stills	1450	1234	Cupido	
1452	Nash	1450	1234	Cupido	
1453	Young	1447	1234	Cupido	
1234	Cupido		1447	Roos	1234
1447	Roos	1234	1447	Roos	1234
1448	Voskuil		1447	Roos	1234
1449	Grunberg		1447	Roos	1234
1450	Crosby		1447	Roos	1234
1451	Stills	1450	1447	Roos	1234
1452	Nash	1450	1447	Roos	1234
1453	Young	1447	1447	Roos	1234
1234	Cupido		1448	Voskuil	
1447	Roos	1234	1448	Voskuil	
...	. ..	...	...	...	...
...	...	...	...	...	...
1452	Cupido		1453	Young	1447
1453	Roos	1234	1453	Young	1447

Figuur 7.18 Cartesisch product van Klant met zichzelf

SQL-statement:

```
select *
from   Klant K, Klant A
```

7.20 SQL-statement:

```
select Aangebracht.nr, Aangebracht.naam, Aanbrenger.nr aanbrengernr,
       Aanbrenger.naam aanbrengernaam
from   Klant Aangebracht
       left outer join Klant Aanbrenger
          on Aangebracht.aanbrenger = Aanbrenger.nr
```

7.21 SQL-statement:

```
select Orl.order_, Orl.artikel, A.omschrijving,
     cast(Orl.aantal*A.verkoopprijs*(1-korting/100.00) as numeric(7,2))
     bedrag
from   Orderregel Orl
       join Artikel A on Orl.artikel = A.nr
       join Kortingsinterval KI on Orl.aantal between
                                   KI.beginaantal and KI.eindaantal
```

Opmerking: in plaats van `cast(… as numeric(7,2))` lijkt ook `round(…, 2)` een goede keuze. Hierbij blijkt echter te worden afgerond op het aantal decimalen van het eerste argument. `Numeric(7,2)` is het datatype van Orderregel.bedrag.

7.22 SQL-statement:

```
select   Kht.nr, K.naam, Ag.omschrijving
from     Klacht Kht
         join Orderregel Orl on Kht.order_ = Orl.order_
                             and Kht.volgnr = Orl.volgnr
         join Order_ O on Orl.order_ = O.nr
         join Klant K on O.klant = K.nr
         join Artikel A on Orl.artikel = A.nr
         left outer join Artikelgroep Ag on A.artikelgroep = Ag.code
where    behandeld = 'N'
order by Kht.order_, Kht.volgnr
```

Uitwerkingen hoofdstuk 8

8.1 Het eenvoudigst is de telling te baseren op de optionele kolom examinator:

```
select count(examinator)
from   Cursus
```

Gaat u uit van `count(*)`, dan is een selectieconditie nodig. Dat is wel duidelijker:

```
select count(*)
from   Cursus
where  examinator is not null
```

In plaats van `count(*)` kunt u ook een verplichte kolom gebruiken, liefst de primaire sleutel: `count(code)`, of een constante, bijvoorbeeld: `count('x')`.

8.2 Het gaat hier om het tellen van een aantal rijen van tabel Cursus. De voorwaarde heeft echter betrekking op tabel Docent. We moeten dus Cursus joinen met Docent en een aantal rijen tellen van die join:

```
select count(*)
from   Cursus C
       join Docent D on C.examinator = D.acr
where  D.naam = 'C.Date'
```

8.3 '`select cursus`' dwingt rijgewijze verwerking af: elke Inschrijving-rij wordt geprojecteerd op kolom cursus en geeft daarbij één resultaatrij.
'`select count(*)`' dwingt statistische verwerking af: één resultaatwaarde van de hele 'tabel als groep'.
Deze beide manieren van verwerken leveren ongelijksoortige resultaten op en zijn niet in één query te combineren. De vraagstelling is niettemin zinvol. In paragraaf 3 zien we hoe het moet.

8.4 SQL-statement:

```
select   examinator, sum(uren)
from     Cursus
where    examinator is not null
group by examinator
```

De `not null`-eis is opgenomen in de `where`-clausule. In een `having`-clausule kan ook, het gaat immers om de groeperingskolom. Er geldt echter: hoe eerder we de omvang van een (tussen)resultaattabel reduceren, hoe beter. Aangezien 'where' voorafgaat aan 'having', kiezen we hier voor 'where'.

8.5 Omdat code als primaire sleutel uniek is, zal elke groep één rij omvatten, zodat elke `count(*)` de waarde 1 oplevert. We krijgen dus als uitvoer een kolom met alleen enen. Groeperen op een primaire sleutel of een andere kolom(combinatie) die uniek is, is niet zinvol.

8.6 Elke groep die als resultaat van groeperen ontstaat, telt één of meer rijen. Lege groepen bestaan niet. Een `count(*)`, in combinatie met `group by`, is dus minimaal 1. Zonder groeperen kan `count(*)` wel 0 zijn, bijvoorbeeld bij een lege tabel, of indien geen enkele rij aan de selectievoorwaarde voldoet, zoals bij:

```
select count(*)
from   Cursus
where  1 = 0
```

8.7a Om docenten die van geen enkele cursus begeleider zijn, niet te missen, is een left outer join nodig.

```
select   D.acr, count(B.docent)
from     Docent D
         left outer join Begeleider B on D.acr = B.docent
group by D.acr
```

Resultaat:

```
ACR        COUNT
====== =========
BAC            2
COD            0
DAT            3
```

Ga na dat bij gebruik van `count(*)` docenten die geen begeleider zijn, een 1 zouden krijgen in plaats van een 0.

b In plaats van op de primaire sleutel D.acr groepeert u nu op de alternatieve sleutel D.naam:

```
select   D.naam, count(B.docent)
from     Docent D
         left outer join Begeleider B on D.acr = B.docent
group by D.naam
```

c Nu moet u verfijnd groeperen op acr en naam, ondanks dat naam functioneel afhankelijk is van acr:

```
select   D.acr, D.naam, count(B.docent)
from     Docent D
         left outer join Begeleider B on D.acr = B.docent
group by D.acr, D.naam
```

8.8 Bij het nemen van de (inner) join verdwijnen niet alleen de kindrijen zonder corresponderende ouderrij, maar ook de ouderrijen zonder corresponderende kindrij. U kunt dus nooit verwachten via een inner join nog iets over die verdwenen ouderrijen te weten te komen. In dit geval: over docenten zonder mentorstudenten.
Verder is het zo dat, wélke verzameling rijen u ook groepeert, een groep altijd één of meer rijen bevat. Lege groepen bestaan niet. Een `count(*)` is dus, bij dit type navigatie, altijd groter dan 0. Daardoor is `count(*) > 0` een loze voorwaarde en staat `count(*) = 0` gelijk aan *false*.

8.9 Neem eerst de volgende query, met een extra kolom S.nr en zonder de `where`-clausule:

```
select   D.naam, S.nr
from     Docent D
         left outer join Student S on D.acr = S.mentor
```

Deze heeft als resultaat:

```
NAAM            NR
========= =========
C.Date            1
C.Date            2
E.Codd            3
C.Bachman    <NULL>
```

De laatste rij is een 'toegevoegde rij' ten gevolge van het nemen van de left outer join. En dat is precies de rij waar het om gaat! De conditie `S.nr is null` selecteert die toegevoegde rij.

8.10 We bepalen het gemiddelde cijfer per cursus (dit vraagt om groeperen van tabel Inschrijving) en nemen van deze gemiddelden het maximum:

```
select   max(avg(cijfer))
from     Inschrijving
group by cursus
-- (geen Firebird)
```

8.11 De volgorde waarin de clausules worden verwerkt, is: eerst `from`, dan `group by`, dan `having` en als laatste `select`. Op het moment dat de `having` wordt verwerkt, heeft de kolomalias aantal_tentamens nog geen betekenis. Het is een kolomnaam van het eindresultaat. De juiste vorm is: `having count(*)` >= 1.

Opmerking: in voorbeeld 8.12 zagen we dat Firebird bij groeperen op een berekende expressie een alias als groeperingsexpressie toestaat, in afwijking van het door ons geschetste conceptuele algoritme.

8.12 SQL-statement:

```
select max(verkoopprijs - inkoopprijs),
       avg(verkoopprijs - inkoopprijs)
from   Artikel
```

8.13 SQL-statement:

```
select   artikelgroep,
         max(verkoopprijs - inkoopprijs),
         avg(verkoopprijs - inkoopprijs)
from     Artikel
group by artikelgroep
```

Hoewel dit niet vanzelfsprekend is, voegt het groeperingsmechanisme rijen waarvoor de groeperingswaarde null is (de losse artikelen in dit geval), samen tot één groep.
Mooier nog is om in de `select`-lijst artikelgroep te vervangen door de functieaanroep `coalesce (artikelgroep, 'losse artikelen')`.

8.14 SQL-statement:

```
select round(avg(Orl.aantal * (A.verkoopprijs * (1-K.korting/100.00)
       - A.inkoopprijs)), 2) gemiddelde_brutowinst
from   Orderregel Orl
       join Artikel A on Orl.artikel = A.nr
       join Kortingsinterval K on Orl.aantal
                                  between K.beginaantal and K.eindaantal
```

Voor het gebruik van 100.00 in plaats van 100: zie opgave 7 van de zelftoets van leereenheid 8. De `round`-functie rondt wel af, maar toont nog extra nullen. U kunt deze afkappen door de uitkomst te casten naar een kleiner datatype, bijvoorbeeld: `cast( … as numeric(5,2))`.

8.15 SQL-statement:

```
select O.nr, O.datum,
       round(sum(Orl.aantal * (A.verkoopprijs * (1-K.korting/100.00)
       - A.inkoopprijs)), 2)
from   Order_ O
       join Orderregel Orl on O.nr = Orl.order_
       join Artikel A on Orl.artikel = A.nr
       join Kortingsinterval K on Orl.aantal
                                  between K.beginaantal and K.eindaantal
group by O.nr, O.datum
```

8.16 SQL-statement:

```
select   Ag.code, Ag.omschrijving
from     Artikelgroep Ag
         join Artikel A on Ag.code = A.artikelgroep
group by Ag.code, Ag.omschrijving
having   count(*) >= 3
```

Dat hier ook op de omschrijving gegroepeerd moet worden, is niet 'logisch': immers de omschrijving is functioneel afhankelijk van de soortcode. Zie paragraaf 4.2 over 'onverwacht verfijnd groeperen'.

8.17 SQL-statement:

```
select   A.nr, A.omschrijving, count(*)
from     Artikel A
         join Orderregel Orl on A.nr = Orl.artikel
         join Klacht Kht on Orl.order_ = Kht.order_
                            and Orl.volgnr = Kht.volgnr
group by A.nr, A.omschrijving
having   count(*) >= 2
```

Ook hier hebben we te maken met 'onverwacht verfijnd groeperen'.

8.18 De query deugt niet vanwege de overbodige `having`-clausule. Na groeperen is elke `count(*)`-waarde immers groter dan 0. De juiste oplossing krijgt u door de `having`-clausule te schrappen.

8.19 SQL-statement:

```
select   avg(count(*))  -- (geen Firebird)
from     Orderregel
group by order_
```

Uitwerkingen hoofdstuk 9

9.1 De linkeroperand van de selectieconditie past bij een rijgewijze verwerking van Inschrijving, terwijl de rechteroperand alleen betekenis heeft voor een groep. We kunnen dus een foutmelding verwachten die vrij vertaald neerkomt op: 'scheve conditie'.

9.2a Oplossing met een left outer join:

```
select   C.code, C.naam, count(I.cursus)
from     Cursus C
         left outer join Inschrijving I on C.code = I.cursus
group by C.code, C.naam
```

b Beide oplossingen zijn goed. In de oplossing met de subselect is de stapsgewijze aanpak zichtbaar. Deze oplossing is daardoor eenvoudiger te begrijpen en conceptueel mooier. Dat is echter niet het enige argument dat telt. Bij een hele grote database moeten we ook rekening houden met de performance. Maar daarover breken we ons pas het hoofd in leereenheid 14 ' Query-optimalisatie'.

9.3 We moeten in de subselect nesting van groepsfuncties vermijden. Dat doen we als volgt, op de manier van voorbeeld 9.3:

```
select   C.code, C.naam, count(*)
from     Cursus C
         join Inschrijving I on C.code = I.cursus
group by C.code, C.naam
having   count(*) =
         (--maximum aantal inschrijvingen per cursus
          select   max(aantal_inschrijvingen)
          from     (select   count(*) aantal_inschrijvingen
                    from     Inschrijving
                    group by cursus))
```

9.4a De subselect verandert niet wezenlijk:

```
select code, naam
from   Cursus
where  code in (-- verzameling codes van cursussen met voorkennis DW
                -- of II
                select cursus
                from   Voorkenniseis
                where  voorkennis in ('DW', 'II'))
```

Desgewenst kan de logische conditie `voorkennis in ('DW', 'II')` vervangen worden door `voorkennis = 'DW' or voorkennis = 'II'`.

b Het ligt voor de hand de join als volgt aan te passen:

```
select C.code, C.naam
from   Cursus C
       join Voorkenniseis V on C.code = V.cursus
where  V.voorkennis in ('DW', 'II')
```

Het resultaat is echter niet wat we willen:

```
CODE   NAAM
====== =========
DB     Databases
DB     Databases
```

Cursus DB wordt meervoudig vermeld omdat we vanuit tabel Cursus navigeren naar Voorkenniseis en bij de ouderrij van DB twee kindrijen vinden.

Een juiste oplossing vinden we door 'distinct' toe te voegen:

```
select distinct C.code, C.naam
from   Cursus C
       join Voorkenniseis V on C.code = V.cursus
where  V.voorkennis in ('DW', 'II')
```

Zie paragraaf 2.3 voor een kritische bespreking van een select-met-distinct bij een vergelijkbaar probleem.

9.5 Oplossing met join:

```
select T.student, T.cursus, T.volgnr, T.datum
from   Tentamen T
       join Inschrijving I on T.student = I.student
                           and T.cursus = I.cursus
where  I.datum >= '01-feb-2012'
```

9.6 Joinoplossing voor voorbeeld 9.8:

```
select   I.student, I.cursus, I.datum
from     Inschrijving I
         join Tentamen T on I.student = T.student
                         and I.cursus = T.cursus
group by I.student, I.cursus, I.datum
having   count(*) >= 2
```

Let op de volgorde van de joinoperanden: deze correspondeert met het navigatiepad. Voor de correcte werking maakt dit niets uit; voor 'het verhaal' dat in elke oplossing wordt verteld, des te meer.

9.7 De selectieconditie is altijd *waar* (gevolg van de referentiële-integriteitsregel) en kan dus worden weggelaten.

9.8a We vinden een oplossing in twee stappen, leidend tot een subselect met `in`. Het commentaar geeft een deelprobleem na de eerste stap.

```
select code, naam, examinator
from   Cursus
where  examinator in (-- verzameling van alle begeleidercodes
                      select docent
                      from   Begeleider)
```

Opmerking: de subselect, indien als zelfstandige query uitgevoerd, kan een docent meervoudig weergeven. Kolom docent is immers niet uniek. Een subselect echter wordt altijd als verzameling opgevat. Een distinct ('`select distinct docent`') is daarom niet nodig.

b De subselect in onderdeel a wordt nu gecorreleerd:

```
select code, naam, examinator
from   Cursus C
where  examinator in (-- verzameling begeleidercodes bij 'die' cursus
                      select docent
                      from   Begeleider
                      where  cursus = C.code)
```

Opmerking: de alias is nu niet nodig. Wanneer we die weglaten, luidt de conditie in de subselect: cursus = Cursus.code. We zullen echter bij een gecorreleerde subselect altijd een alias gebruiken.

9.9 De subselect is niet-gecorreleerd, dus voor elke rij van de hoofdselect hetzelfde. Het resultaat ervan is wel of niet een lege verzameling. Indien leeg, geeft de exists een *false*. Indien niet-leeg geeft de subselect een *true*. Voor de query als geheel zijn er dus maar twee mogelijkheden: hij geeft geen enkele cursus of juist alle cursussen.

9.10 Ja, dat is het geval, als volgt.

```
select acr, naam
from   Docent
where  -- docent hoort tot begeleiders van DW
       acr in (select docent
               from   Begeleider
               where  cursus = 'DW')
       and
       -- docent hoort tot begeleiders van IM
       acr in (select docent
               from   Begeleider
               where  cursus = 'IM')
```

U ziet: de oplossing wordt er zelfs eenvoudiger door.

9.11a Oplossing met left outer join:

```
select C.code, C.naam
from   Cursus C
       left outer join Inschrijving I on C.code = I.cursus
                                         and vrijstelling = 'J'
where  I.student is null
```

Toelichting: Dit is een bijzondere join: er wordt gejoind over sleutelgelijkheid, maar ook over de extra voorwaarde vrijstelling = 'J'. De 'left' zorgt dat cursussen die niet aan de joinconditie voldoen, toch eenmalig worden opgenomen, met nulls in de I-kolommen. Het is juist om die cursussen te doen.

b Een oplossing met statistische subselect:

```
select code, naam
from   Cursus C
where  0 = (select count(*)
            from   Inschrijving
            where  cursus = C.code
                   and vrijstelling = 'J')
```

Opmerking: de conditie `0 = (`***subselect***`)` is logisch gesproken gelijkwaardig aan `(`***subselect***`) = 0`. De tweede vorm is echter niet in elk SQL-dialect toegestaan.

9.12 Oplossing met `not in`:

```
select acr, naam
from   Docent
where  acr not in (select mentor
                   from   Student
                   where  mentor is not null)
```

Resultaat:

```
ACR    NAAM
====== ======
COD    E.Codd
```

Opmerking: zonder de conditie 'mentor is not null' zou de waardenverzameling van de subselect ook een null bevatten. De where-conditie van de hoofdselect kan dan nooit *true* worden.

Die complicatie doet zich niet voor bij `not exists`:

```
select acr, naam
from   Docent D
where  not exists (select *
                   from   Student
                   where  mentor = D.acr)
```

Er is geen oplossing met een inner join van Docent en Student: bij de rij-voor-rij-verwerking daarvan kun je nooit per rij concluderen dat de betreffende docent voldoet. Met een left outer join lukt het wel:

```
select D.acr, D.naam
from   Docent D
       left outer join Student S on D.acr = S.mentor
where  S.nr is null
```

9.13 Eerste stap:

```
select nr, naam
from   Student
where  not exists
       (cursussen waarvoor geen inschrijving bestaat voor 'deze' student,
        met een vrijstelling of een voldoende)
```

Tweede stap:

```
select nr, naam
from   Student
where  not exists
       (-- cursussen waarvoor geen inschrijving bestaat voor 'deze'
        -- student, met een vrijstelling of een voldoende
        select *
        from   Cursus
        where  not exists
               (inschrijving van 'deze' student voor 'deze' cursus met een vrijstelling
                of een voldoende))
```

In de derde en laatste stap verwijzen we daadwerkelijk naar ‘deze student’ en naar ‘deze cursus’:

```
select nr, naam
from   Student S
where  not exists
       (-- cursussen waarvoor geen inschrijving bestaat voor 'deze'
        -- student, met een vrijstelling of een voldoende
        select *
        from   Cursus C
        where  not exists
               (-- inschrijving van 'deze' student voor 'deze' cursus
                -- met een vrijstelling of een voldoende
                select *
                from   Inschrijving
                where  student = S.nr
                       and cursus = C.code
                       and (vrijstelling = 'J' or cijfer >= 6)))
```

9.14 Een alternatief voor de query met `all` (voorbeeld 9.21) gebruikt de functie `min`:

```
select student, cursus, volgnr, datum, cijfer
from   Tentamen T
where  volgnr >= 2
       and cijfer < (-- minimumcijfer van voorgaande pogingen
                     select min(cijfer)
                     from   Tentamen Voorgaande
                     where  Voorgaande.student = T.student
                            and Voorgaande.cursus = T.cursus
                            and Voorgaande.volgnr < T.volgnr)
```

Een alternatief voor de query met any (voorbeeld 10.22) gebruikt de functie max:

```
select student, cursus, volgnr, datum, cijfer
from   Tentamen T
where  volgnr >= 2
       and cijfer < (-- maximumcijfer van voorgaande pogingen
                     select max(cijfer)
                     from   Tentamen Voorgaande
                     where  Voorgaande.student = T.student
                            and Voorgaande.cursus = T.cursus
                            and Voorgaande.volgnr < T.volgnr)
```

9.15 We beginnen met een eenvoudige, maar belangrijke stap:

```
select   code, naam, aantal_inschrijvingen
from     vCursus
where    'deze' cursus zit in de top-3
order by aantal_inschrijvingen desc
```

Vervolgens vertalen het top-3 criterium naar een formulering waarin de betekenis van 'top-3' tot uiting komt:

```
select   code, naam, aantal_inschrijvingen
from     vCursus
where    3 > (het aantal cursussen met meer inschrijvingen dan 'deze' cursus)
order by aantal_inschrijvingen desc
```

waarna we als eindoplossing noteren:

```
select code, naam, aantal_inschrijvingen
from   vCursus vC
where  3 > (select count(*)
            from   vCursus
            where  aantal_inschrijvingen > vC.aantal_inschrijvingen)
order by aantal_inschrijvingen desc
```

9.16 SQL-statement:

```
select nr, datum
from   Order_
where  datum = (select min(datum)
                from   Order_)
```

9.17 SQL-statement:

```
-- geen Firebird
select   Ag.omschrijving
from     Artikelgroep Ag
         join Artikel A on Ag.code = A.artikelgroep
group by Ag.omschrijving
having   count(*) = (select   max(count(*))
                     from     Artikel
                     where    artikelgroep is not null
                     group by artikelgroep)
```

Zonder geneste groepsfuncties:

```
select   Ag.omschrijving
from     Artikelgroep Ag
         join Artikel A on Ag.code = A.artikelgroep
group by Ag.omschrijving
having   count(*) = (select max(aantal_artikelen)
                     from   (select   count(*) aantal_artikelen
                             from     Artikel
                             where    artikelgroep is not null
                             group by artikelgroep))
```

9.18 SQL-statement:

```
select nr, omschrijving, artikelgroep
from   Artikel A
where  verkoopprijs - inkoopprijs =
       (select max(verkoopprijs - inkoopprijs)
        from   Artikel
        where  artikelgroep = A.artikelgroep)
```

9.19 Dit probleem leent zich goed voor een oplossing met een subselect in de select-clausule:

```
select nr, omschrijving, (select sum(bedrag)
                          from   Orderregel
                          where  artikel = A.nr)
from   Artikel A
```

Deze oplossing geeft echter nulls voor een 'lege som', dat is bij artikelen zonder orderregels. We geven de subselect daarom mee als argument aan de null-vervangfunctie `coalesce` (dat mag!). Dit geeft:

```
select nr, omschrijving, coalesce((select sum(bedrag)
                                   from   Orderregel
                                   where  artikel = A.nr), 0)
from   Artikel A
```

Het kan ook met een left outer join:

```
select   A.nr, A.omschrijving, coalesce(sum(bedrag), 0)
from     Artikel A
         left outer join Orderregel Orl on A.nr = Orl.artikel
group by A.nr, A.omschrijving
```

9.20 Eerste poging:

```
select nr, naam
from   Klant
where  nr not in (-- klantaanbrengers
                  select aanbrenger
                  from   Klant)
```

Dit geeft verrassenderwijs: ' no records returned'. De fout blijkt snel wanneer we de subselect uitvoeren als zelfstandig statement: het resultaat bevat een aantal null' s! Dat maakt dat elke conditie 'nr not in (…)' de waarde *unknown* krijgt. Een juiste oplossing is:

```
select nr, naam
from   Klant
where  nr not in (-- klantaanbrengers
                  select aanbrenger
                  from   Klant
                  where  aanbrenger is not null)
```

Een gecorreleerde subselect met `not exists` is hier ook een mogelijkheid:

```
select nr, naam
from   Klant Aanbrenger
where  not exists (-- klantaanbrengers
                   select aanbrenger
                   from   Klant Aangebracht
                   where  aanbrenger = Aanbrenger.nr)
```

9.21 Twee oplossingen met een subselect:

```
select nr, naam
from   Klant
where  nr not in (select klant
                  from   Order_)

select nr, naam
from   Klant K
where  not exists (select *
                   from   Order_
                   where  klant = K.nr)
```

Het kan (wat vergezocht maar instructief) ook met een left outer join, waarbij de rijen die door de left outer join erbij komen ten opzichte van de inner join, de enige zijn die de selectieconditie overleven:

```
select K.nr, K.naam
from   Klant K
       left outer join Order_ O on K.nr = O.klant
where  O.klant is null
```

9.22 Er zijn vele oplossingen mogelijk. Om te beginnen volgt hier een oplossing met een geneste subselect met `in`. Omdat een aantal artikelen wordt gevraagd, starten we de navigatie vanuit de tabel Artikel:

```
select count(*)
from   Artikel
where  nr in
       (-- artikelnummers van orderregels waarover een klacht bestaat
        select artikel
        from   Orderregel
        where  (order_, volgnr) in (select order_, volgnr
                                    from   Klacht))
```

(Pas deze in Firebird aan, met de 'concatenatietruc')

Voor een eerste variant vervangen we de subselect met `in` door een (gecorreleerde) subselect met `exists`:

```
select count(*)
from   Artikel
where  nr in
       (-- artikelnummers van orderregels waarbij een klacht bestaat
        select artikel
        from   Orderregel Orl
        where  exists (-- klachten bij 'deze' orderregel
                       select *
                       from   Klacht
                       where  order_ = Orl.order_
                              and volgnr = Orl.volgnr))
```

Wie bij deze oplossingen opmerkt dat we ook in Orderregel kunnen starten omdat van de artikelen alleen de artikelnummers van belang zijn, heeft gelijk. Echter, omdat een artikel in meerdere klachten kan voorkomen, is nu een `distinct` vereist, zoals vaker wanneer we de navigatie starten in de 'conceptueel verkeerde tabel':

```
select count(distinct artikel)
from   Orderregel Orl
       where exists (-- klachten bij 'deze' orderregel
                     select *
                     from   Klacht
                     where  order_ = Orl.order_
                            and volgnr = Orl.volgnr)
```

Tot slot geven we een oplossing door alleen te joinen:

```
select count(distinct artikel)
from   Orderregel Orl
       join Klacht Kht on Orl.order_ = Kht.order_
                      and Orl.volgnr = Kht.volgnr
```

9.23 Het gaat om een statistisch overzicht per artikelgroep, dus start het navigatiepad in Artikelgroep. Omdat alleen de groepsomschrijving wordt gevraagd, volstaat het te groeperen op de alternatieve sleutel omschrijving:

```
select   Ag.omschrijving, count(*)
from     Artikelgroep Ag
         join Artikel A on Ag.code = A.artikelgroep
where    -- er bestaat een klacht over het artikel
         A.nr in
         (-- artikelnummers van artikelen waarover een klacht bestaat
          select artikel
          from   Orderregel Orl
                 join Klacht Kht on Orl.order_= Kht.order_
                                and Orl.volgnr = Kht.volgnr)
group by Ag.omschrijving
```

Voor de `where`-clausule bestaan verschillende alternatieven, analoog aan de varianten van opgave 9.21.

9.24 Eerste stap:

```
select nr, naam
from   Klant
where  'deze' klant heeft minstens één order geplaatst
       and
        er zijn geen orders van 'deze' klant waarover geen klacht is ingediend
```

Eerste uitwerking in termen van (`not`) `exists`:

```
select nr, naam
from   Klant
where  exists (verzameling orders van 'deze' klant)
       and not exists
          (verzameling orders van 'deze' klant waarover geen klacht is ingediend)
```

De deelproblemen in natuurlijke taal laten zich gemakkelijk vertalen in subselects:

```
select nr, naam
from   Klant K
where  exists (-- verzameling orders van 'deze' klant
               select *
               from   Order_
               where  klant = K.nr)
       and not exists
          (-- verzameling orders van 'deze' klant waarover geen klacht
           -- is ingediend
           select *
           from   Order_ O
           where  klant = K.nr
                  and not exists
                      (-- de verzameling klachten bij 'deze' order
                       select *
                       from   Klacht
                       where  order_ = O.nr))
```

Merk op dat we bij navigatie van Order_ naar Klacht als het ware over Orderregel heenspringen. Dit is mogelijk omdat Orderregel hier slechts de waarden van de sleutel (order_) overdraagt.

9.25 Een oplossing met geneste subselects:

```
select nr, naam
from   Klant
where  nr in (-- verzameling klantnummers van klanten die
              -- een slevel of een wigbek hebben gekocht
              select klant
              from   Order_
              where  nr in (select order_
                            from   Orderregel
                            where  artikel in
                                   (select nr
                                    from   Artikel
                                    where  omschrijving = 'slevel'
                                           or omschrijving = 'wigbek')))
```

De laatste conditie kan ook als volgt geformuleerd worden: `omschrijving in ('slevel', 'wigbek')`.
Deze oplossing laat fraai laat zien hoe we door subselectnavigatie van tabel naar tabel kunnen wandelen. Joinnavigatie geeft een wat compactere oplossing:

```
select nr, naam
from   Klant
where  nr in (-- verzameling klantnummers van klanten die
              -- een slevel of een wigbek hebben gekocht
              select klant
              from   Order_ O
                     join Orderregel Orl on O.nr = Orl.order_
                     join Artikel A on Orl.artikel = A.nr
              where  omschrijving = 'slevel'
                     or omschrijving = 'wigbek')
```

We zeggen ‘compacter’ en niet ‘eenvoudiger’. In wezen gebeurt er immers hetzelfde. Overigens kan ook de Klant-tabel in de join worden opgenomen, ten koste van een `distinct` in de `select`-clausule. Dit ‘pseudo-groeperen’ vinden we echter niet zo fraai.

9.26 Deze opgave lijkt oppervlakkig gezien erg op de vorige. Het is daarom verleidelijk de `or` in de laatste regel te vervangen door `and`. De conditie `omschrijving = 'slevel' and omschrijving = 'wigbek'` is echter altijd *false*. Dat gaat dus zo niet.

De volgende oplossing is een van de vele mogelijkheden. Door een stapsgewijze oplossing (de subselects eerst opstellen op het niveau van de commentaren) wordt het probleem opgebroken in eenvoudige deelproblemen:

```
select nr, naam
from   Klant K
where  nr in (-- verzameling klantnummers van klanten die
              -- een slevel hebben gekocht
              select klant
              from   Order_
              where  nr in
                     (select order_
                      from   Orderregel
                      where  artikel in
                             (select nr
                              from   Artikel
                              where  omschrijving = 'slevel')))
       and
       nr in (-- verzameling klantnummers van klanten die
              -- een wigbek hebben gekocht
              select klant
              from   Order_
              where  nr in
                     (select order_
                      from   Orderregel
                      where  artikel in
                             (select nr
                              from   Artikel
                              where  omschrijving = 'wigbek')))
```

9.27 Eerst definiëren we een view vArtikelomzet:

```
create view vArtikelomzet (nr, omschrijving, omzet)
as
select nr, omschrijving, coalesce((select sum(bedrag)
                                   from   Orderregel
                                   where  artikel = A.nr), 0)
from   Artikel A
```

Hierna is het probleem gereduceerd tot een standaard Top-*n*-probleem, met als oplossing:

```
select   nr, omschrijving, omzet
from     vArtikelomzet vA
where    5 > (select count(*)
              from   vArtikelomzet
              where  omzet > vA.omzet)
order by omzet desc
```

Uitwerkingen hoofdstuk 10

10.1b Invoeren van de cursus:

```
insert into Cursus (code, naam, uren, credits)
values ('BI', 'Business Intelligence', 120, 4);
```

De examinator wordt null. Dat kan ook expliciet gebeuren, zoals in OpenSchoolInsert.sql.

Invoeren van voorkenniseisen:

```
insert into Voorkennis values ('BI', 'DB');
insert into Voorkennis values ('BI', 'IM')
```

10.2 Geen uitwerking.

10.3 Eén statement is voldoende:

```
delete from Cursus
where code = 'BI'
```

Vanwege de cascading delete worden de bijbehorende rijen van Voorkenniseis meeverwijderd.

10.4 Een oplossing is een delete met een gecorreleerde subselect

```
delete from Tentamen T
where exists (select *
              from   Tentamen
              where  cursus = T.cursus
                     and student = T.student
                     and volgnr > T.volgnr
                     and cijfer > T.cijfer)
```

De subselect met `exists` kan vervangen worden door een subselect met `in`.

10.5 SQL-statement:

```
delete from Docent
where  acr not in (select mentor
                   from   Student
                   where  mentor is not null)
       and acr not in (select examinator
                       from   Cursus)
                       where  examinator is not null)
       and acr not in (select docent
                       from   Begeleider)
       and acr not in (select vervanger
                       from   Docent)
                       where  vervanger is not null)
```

De subselects kunnen uiteraard worden vervangen door gecorreleerde varianten met `not exists`.

10.6 Het gevraagde update-statement bevat een gecorreleerde subselect:

```
update Inschrijving I
set    cijfer = (select max(cijfer)
                 from   Tentamen
                 where  student = I.student
                        and cursus = I.cursus
                        and cijfer is not null)
where  cursus in (select cursus
                  from   Tentamen)
```

De subselect met `in` kan uiteraard vervangen worden door een (gecorreleerde) subselect met `exists`. Ga na dat de laatste `where`-clausule ook achterwege mag blijven. Het is echter wel netjes om de berekening alleen uit te voeren voor cursussen waarvoor een tentamen bestaat.

10.7 We gebruiken het volgende trucje: verhoog alle nummers eerst met een 'veilige' waarde en verlaag ze daarna met die veilige waarde minus 1. Bijvoorbeeld:

```
update Student
set    nr = nr + 1000;
update Student
set    nr = nr - 999;
```

10.8 Voldoende is de kolommen nr en klant te vullen:

```
insert into Order_ (nr, klant)
values (5900, 1447)
```

De kolomwaarde van datum wordt de systeemdatum (`sysdate()`, zie create-script) en totaalbedrag blijft leeg.

10.9a Insert-script:

```
insert into ArtikelExtra values (501, 'keggebek',   16.50, 120);
insert into ArtikelExtra values (502, 'handboom',   85.00,  75);
insert into ArtikelExtra values (503, 'radiobutton', 6.50,  28);
insert into ArtikelExtra values (504, 'zwalik',     24.00, 185);
insert into ArtikelExtra values (505, 'slewel',     14.75,  10);
insert into ArtikelExtra values (506, 'wormgat',    99.00,   7);

insert into Correspondentie values (449, 501);
insert into Correspondentie values (180, 504);
insert into Correspondentie values (351, 505);
commit
```

c Het gaat in twee stappen.
Stap 1: aanpassen van omschrijvingen en voorraden in Artikel aan die van corresponderende rijen in ArtikelExtra
Stap 2: invoeren in Artikel van de artikelen in ArtikelExtra die niet in Artikel (en dus niet in Correspondentie) voorkomen.

Stap 1

```
update Artikel A
set    omschrijving = (select AE.naam
                       from   ArtikelExtra AE
                              join Correspondentie C on AE.anr = C.anr
                       where  C.nr = A.nr),
       voorraad = voorraad +
                  (select AE.aantal
                   from   ArtikelExtra AE
                          join Correspondentie C on AE.anr = C.anr
                   where  C.nr = A.nr)
where  nr in (select nr
              from   Correspondentie)
```

Stap 2

```
insert into Artikel (nr, omschrijving, verkoopprijs, inkoopprijs,
                     voorraad)
   select anr, naam, round(prijs * 1.30, 2), prijs, aantal
   from   ArtikelExtra
   where  anr not in (select anr
                      from   Correspondentie)
```

10.10 Onderstaand stappenplan gaat ervan uit dat de voorbeeldpopulatie een goede testpopulatie is, met voldoende illustratieve gegevens voor te verwijderen klanten. Desgewenst kunt u de grensdatum 1 januari 2007 veranderen in een meer recente datum, zodat de voorbeeldpopulatie voldoende illustratief wordt.

Stap 1: Verwijder alle klachten van de klanten die sinds 2007 niets meer hebben besteld.
Stap 2: Verwijder alle orders met orderregels (cascading delete!) van de klanten die sinds 2007 niets meer hebben besteld.
Stap 3: Maak een aanbrengernummer null, indien dit het klantnummer is van een klant zonder order.
Stap 4: Verwijder alle klanten zonder order.

Het achterwege laten van stap 3 zou problemen geven bij stap 4. Een klant kan immers niet worden verwijderd indien er nog een verwijzing naar bestaat.

Stap 1

```
delete
from  Klacht
where order_ in
      (select O.nr
       from   Order_ O
       where  -- de klant bij 'deze' order heeft geen order van 2007 of
              -- later
              not exists (select *
                          from   Order_
                          where  klant = O.klant
                                 and datum >= '01-jan-2007'))
```

Stap 2

```
delete
from  Order_ O
where -- de klant bij deze order heeft geen order van 2007 of later
      not exists (select *
                  from   Order_
                  where  klant = O.klant
                         and datum >= '01-jan-2007')
```

Nu is de weg vrij om ook die klanten zelf te verwijderen. Omdat Klant echter een recursieve verwijzing bevat, moeten vooraf de verwijzingen naar die klanten null gemaakt worden.

Stap 3

```
update Klant
set    aanbrenger = null
where  -- aanbrenger is het klantnummer van een klant zonder order
       aanbrenger not in (select klant
                          from Order_)
```

Stap 4

```
delete
from  Klant
where -- bij 'deze' klant hoort geen enkele order
      nr not in (select klant
                 from   Order_)
```

Uitwerkingen hoofdstuk 11

11.1 Creëren testdatabase:

```
create database 'Testdatabase.fdb' user 'Sysdba' password 'masterkey'
```

11.2 Beëindigen van een sessie gaat met `disconnect`. Een poging om daarna een tabel aan te maken, leidt tot de foutmelding: 'invalid database handle (no active connection)'.

11.3 Na het beëindigen van de huidige sessie (zie vorige opgave) past u de naam aan via het besturingssysteem van de computer.

11.4 Verwijderen van Testbase.fdb (mits dit de actieve database is):

```
drop database
```

Alternatief: verwijderen via het besturingssysteem. Er is nóg een mogelijkheid: verwijderen via het databasemenu van de Boekverkenner.

11.5 Het volgende stappenplan volgt min of meer de onderdelen a en b van de opgave:

a Kolom Reis.transport moet vervoer gaan heten:
   1 Verwijder foreign key fk_reis_met_transport
   2 Maak nieuwe kolom Reis.vervoer
   3 Kopieer inhoud van Reis.transport naar Reis.vervoer
   4 Verwijder kolom Reis.transport

b Tabel Transport moet Vervoer gaan heten:
   1 Maak nieuwe tabel Vervoer
   2 Hevel inhoud van Transport over naar Vervoer
   3 Verwijder tabel Transport

c Breng de in stap 1a verwijderde foreign key opnieuw aan (aangepast).

Een script hierbij:

```
--a
alter table Reis
drop constraint fk_reis_met_transport;
alter table Reis
add vervoer varchar(2) not null;
update Reis
set    vervoer = transport;
alter table Reis
drop transport;

--b
create table Vervoer
(code           varchar(2)     not null,
 omschrijving   varchar(12)    not null,
 constraint pk_vervoer primary key (code),
 constraint un_vervoeromschrijving unique (omschrijving)
);
insert into Vervoer
   select *
   from   Transport;
drop table Transport;

--c
alter table Reis
add constraint fk_reis_met_vervoer
    foreign key (vervoer) references Vervoer(code)
```

Opmerking: het hernoemen van kolommen hebben we vermeden, hoewel dat in Firebird in principe mogelijk is. Het stuit echter snel op problemen met 'afhankelijkheden' in de data dictionary.

11.6 De oplossing is: verwijder de verwijzende constraint, hoog in Deelname en in Klant 'ongestoord' de nummers op en maak de constraint weer aan:

```
alter table Deelname
drop constraint fk_deelname_door_klant;
```

Bij een foutmelding: voer eerst een `reconnect` uit.

```
update Klant
set    nr = nr + 1000;

update Deelname
set    klant = klant + 1000;

commit;

alter table Deelname
add constraint fk_deelname_door_klant
    foreign key (klant) references Klant(nr)
       on update cascade
```

Eenvoudiger zou zijn de constraint tijdelijk buiten werking te stellen (*disable*) en later weer actief te maken (*enable*). Dit wordt echter niet ondersteund.

11.7 Het aangepaste `create table`-statement van Reis luidt aldus:

```
create table Reis
(nr              integer      not null,
 vertrekdatum    date         not null,
 duur            integer,
 prijs           numeric(5,2),
 constraint pk_reis primary key (nr),
 constraint ch_nr check (nr between 1 and 2000),
 constraint ch_duur check (not(prijs is not null and duur is null))
)
```

Toelichting

- De `check`-constraint ch_nr controleert of het reisnummer, indien ingevuld, in het aangegeven interval ligt.
- De tweede `check`-constraint luidt, vrij vertaald: `check (not` *verboden toestand*). Een rij verkeert in de verboden toestand wanneer zijn prijs bekend is, maar zijn reisduur niet. Ga na dat het ook wat korter kan: `check (prijs is null or duur is not null)`.

11.8 Geen uitwerking.

11.9 We voegen de `check`-constraint toe via het volgende `alter table`-commando:

```
alter table Deelname
add constraint ch_maximaal_aantal_deelnemers
check (4 > (select count(*)
            from   Deelname
            where  reis = new.reis))
```

Proberen we hierna aan reis 31 een vijfde deelnemer toe te voegen:

```
insert into Deelname values (31, 125)
```

Dan resulteert dit in een foutmelding:

```
Operation violates CHECK constraint CH_MAXIMAAL_AANTAL_DEELNEMERS on view or table DEELNEMER
At trigger 'CHECK_22'.
```

Voor de liefhebbers volgt hier nog een andere oplossing:

```
alter table Deelname
add constraint ch_maximaal_aantal_deelnemers
    check (4 > (select count(*)
                from   Deelname D
                where  D.reis = Deelname.reis))
```

Toelichting: Deelname.reis is gecorreleerd aan 'Deelname' in de eerste regel en is het reisnummer van de 'reis in kwestie'. Voor de brontabel van de `select`-expressie moeten we nu, ter onderscheid, een alias gebruiken (D).

11.10 Geen uitwerking.

11.11a SQL-statements:

```
create sequence sReisnr;
alter sequence sReisnr restart with 100
```

b Uitgaande van de oude tabelnaam Reis:

```
update Reis
set    nr = next value for sReisnr
```

(Controleren in Reis, Bezoek en Deelname.)

11.12 SQL-statement:

```
alter table Artikel
add constraint ch_verkoopprijs check (verkoopprijs > inkoopprijs)
```

11.13a Een eenvoudige manier om een constraintnaam te achterhalen, is bewuste overtreding van de constraint. De foutmelding bevat de constraintnaam.

b We zoeken de naam op door een foutmelding te forceren, bijvoorbeeld door een update waarbij we proberen een artikel een niet-bestaande artikelgroep te geven. Vervolgens verwijderen we die constraint:

```
alter table Artikel
drop constraint INTEG_21
```

Het kan zijn dat bij u de constraint een ander volgnummer heeft. De hoofdletters '`INTEG`' mogen ook kleine letters zijn; deze worden, net als kleine letters in bijvoorbeeld tabel- en kolomnamen, intern in hoofdletters omgezet.

c De actie uit onderdeel a moet nu wel lukken! Na controle (en een rollback) voegt u de verwijssleutel weer toe:

```
alter table Artikel
add foreign key (artikelgroep) references Artikelgroep(code)
```

11.14 *Knelpunten*
Omdat er geen `rename`-commando is, is een nieuwe tabel Client onvermijdelijk. Voor het toevoegen van voorletters en eventueel een voorvoegsel aan de oude klantnamen zijn er meerdere mogelijkheden. Bijvoorbeeld:

1 Klant laten zoals die is en, na overzetten van de populatie, de updates op voorletters en voorvoegsel uitvoeren in Client
2 de correcte en volledige namen van de oude klanten eerst invoeren in Klant en dan pas de populatie overzetten naar Client.

Oplossing 1 stuit op het probleem van de verplichte kolom Client.voorletters: u kunt de populatie alléén overzetten als u ofwel deze kolom vult met tijdelijke waarden, ofwel deze kolom eerst optioneel maakt en achteraf, via een structuurwijziging, alsnog verplicht maakt.
Bij oplossing 2 is een structuurwijzing onvermijdelijk, maar vermijden we het probleem van de verplichte voorletters.
In het hierna volgende stappenplan is voor oplossing 1 gekozen.

Een knelpunt is nog, zoals vaak, de recursieve verwijzing. U kunt immers geen klant toevoegen met een verwijzing naar een andere, nog niet toegevoegde klant. Het eenvoudigst is de aanbrenger-kolom van Client eerst te vullen en pas daarna de recursieve verwijzing toe te voegen.

Stappenplan

1 Creëer Client, zonder de recursieve aanbrengerverwijzing.
2 Kopieer de populatie van Klant naar Client; vul daarbij de verplichte kolom Client.voorletters met een lege string.
3 Breng in Client de recursieve aanbrengerverwijzing aan.
4 Maak nieuwe kolom Order_.client.
5 Vul Order_.client vanuit Order_.klant en commit.
6 Leg een verwijssleutel van Order_.client naar Client.nr.
7 Verwijder de oude verwijssleutel van Order_ naar Klant.
8 Verwijder kolom Order_.klant.
9 Verwijder de recursieve verwijzing in Klant en daarna Klant zelf.

Uitvoering stappenplan

Een werkend script, genaamd OrderdatabaseCConversie.sql, vindt u in de standaard scriptmap (zie 'RelSQL' binnen het Windows Startmenu).

Uitwerkingen hoofdstuk 12

12.1 Geen uitwerking.

12.2a t/m c Zie tekst.

d Log in als Tom en geef commando drop user Test1.

12.3 Per `grant`-statement kunnen rechten worden verleend op één object.

12.4 SQL-statement:

```
revoke update(reisduur) on Reis from Sofie
```

12.5 SQL-statement:

```
grant select on Hemelobject to Sofie, public;
revoke select on Hemelobject from public
```

Sofie behoudt haar privé-leesrecht.

12.6b U krijgt de melding 'no permission for references access to `table` Reis'. U probeert immers een verwijssleutel aan te maken met Reis als 'doeltabel', maar u mist hiervoor het `references`-privilege.

12.7 SQL-statement:

```
revoke all on Reis from Tom, Luc, Sofie, Jip, Lisa;
revoke all on Hemelobject from Tom, Luc, Sofie, Jip, Lisa;
revoke all on Bezoek from Tom, Luc, Sofie, Jip, Lisa;
revoke all on Klant from Tom, Luc, Sofie, Jip, Lisa;
revoke all on Deelname from Tom, Luc, Sofie, Jip, Lisa
```

12.8a SQL-statement (gegeven door gebruiker Ruimtereisbureau):

```
create view vPlaneet (naam)
as select naam
   from   Hemelobject
   where  moederobject = 'Zon'
```

De view vPlaneet is updatable, omdat hij alle verplichte kolommen omvat van één tabel (Hemelobject) daarnaast geen join, `distinct`, `group by` of statistische functie bevat.

b SQL-statements:

```
grant all on vPlaneet to Luc;
grant references on Hemelobject to Luc
```

c SQL-statements:

```
connect 'Ruimtereisbureau.fdb' user 'Luc' password 'pw';
update vPlaneet
set    naam = 'Tranendal'
where  naam = 'Aarde'
```

d SQL-statements:

```
insert into vPlaneet values('Cosimo');
commit
```

In de onderliggende tabel Hemelobject is hierdoor een rij ('Cosimo', `null`, `null`, `null`) ingevoerd. Deze voldoet *niet* aan de selectieconditie van de view en is daardoor via een `select`-opdracht op de view niet zichtbaar. (De `commit` op dit punt is van belang indien u wilt switchen naar gebruiker Ruimtereisbureau.)

e Om zichtbaar te zijn via de view, moet de nieuwe rij in Hemelobject voor moederobject als waarde 'Zon' hebben. Een (niet zo mooie) oplossing is de view een tweede kolom te geven:

```
create view vPlaneet (naam, moederobject)
as select naam, moederobject
   from   Hemelobject
   where  moederobject = 'Zon'
with check option
```

Door de toevoeging `with check option` zal het niet lukken aan moederobject een andere waarde te geven dan 'Zon'. Mooier is een oplossing met een trigger op vPlaneet, maar daarvoor moeten we wachten tot leereenheid 16 'Triggers en stored procedures'.

12.9 vDeelnemer is niet updatable, want zijn query is gebaseerd op een join. Ook vReisFinancieel is niet updatable, want de verplichte kolommen vertrekdatum en transport ontbreken. vReisPlanning is wel updatable.

12.10 Top3 is read-only. De privileges `insert`, `delete` en `update` zijn dus niet van toepassing. Firebird accepteert ze toch en slaat ze op in de data dictionary alsof er niets aan de hand is. Maar probeert u van deze 'rechten' (op iets onmogelijks) gebruik te maken, dan volgt natuurlijk alsnog een foutmelding.

12.11 SQL-statement:

```
create view vKlant
as select naam, geboortedatum
   from   Klant K
```

Updates lukken: doordat de eerste kolom van vKlant een alternatieve sleutel van Klant is, zijn vKlant-rijen eenduidig te herleiden tot Klant-rijen.

Om dezelfde reden kunt u ook deletes uitvoeren, maar alleen voor klanten die niet voorkomen in Deelname. Er is immers ook nog de referentiële-integriteitsregel.
Inserts lukken niet: we missen de verplichte kolom (en primaire sleutel) nr.

12.12 SQL-script:

```
/* Autorisatiescript Ruimtereisbureau
*/

create role rPlanning;
grant all on Transport to rPlanning;
grant all on vReisplan to rPlanning;
grant all on Hemelobject to rPlanning;
grant all on Bezoek to rPlanning;
grant rPlanning to Luc, Lisa;

create role rVerkoop;
grant select on Reis to rVerkoop;
grant select on Hemelobject to rVerkoop;
grant select on Bezoek to rVerkoop;
grant all on Klant to rVerkoop;
grant all on Deelname to rVerkoop;
grant rVerkoop to Jip, Sofie, Lisa;

create role rManagement;
grant select, update(prijs) on Reis to rManagement;
grant select on Hemelobject to rManagement;
grant select on Bezoek to rManagement;
grant select on vTop3 to rManagement;
grant rManagement to Tom
```

12.13 Geen terugkoppeling.

12.14a SQL-statement:

```
create view vKlacht (klachtnr, ordernr, orderregelnr, behandeld,
                     klantnr, klantnaam, artikelnr, artikelomschrijving)
as select Kht.nr, Kht.order_, Kht.volgnr, Kht.behandeld,
          O.klant, K.naam, Orl.artikel, A.omschrijving
   from   Klacht Kht
          join Orderregel Orl on Kht.order_ = Orl.order_
                              and Kht.volgnr = Orl.volgnr
          join Order_ O on Orl.order_ = O.nr
          join Klant K on O.klant = K.nr
          join Artikel A on Orl.artikel = A.nr
   where  behandeld = 'N';
```

b SQL-statement:

```
create view vKlant (klantnr, klantnaam)
as select nr, naam
   from Klant
```

12.15a vKlant is updatable, daar deze aan de volgende drie voorwaarden voldoet: de viewkolommen behoren tot één onderliggende tabel, de view omvat alle verplichte kolommen van die onderliggende tabel en de `select`-expressie bevat geen join, `distinct` of `group by`.

b vKlacht is om twee redenen niet updatable: de viewkolommen zijn niet afkomstig uit één onderliggende tabel en de select-expressie bevat een join.

c Om de kolom 'behandeld' te updaten via de niet-updatable view vKlacht, zouden we een before-insert trigger moeten schrijven die reageert op een insert-poging op vKlacht. We laten de trigger de update uitvoeren.

12.16 Een verfijnd onderscheid is op zijn plaats omdat er, conceptueel gezien, verschillende condities in het geding zijn:
- voor 'insertable' is het voldoende dat de view alle verplichte kolommen van de tabel bevat (dus, uitgaande van Codd-relationaliteit, inclusief de primaire sleutel)
- voor 'deletable' is identificatie voldoende, dus de aanwezigheid van een (primaire of alternatieve) sleutel
- voor 'updatable' (in een nieuwe, beperkte betekenis) is identificatie voldoende (zie deletable) tezamen met de aanwezigheid van de te updaten kolom(men).

12.17 Alles in één script:

```
create role rArtikelbeheer;
grant all on Artikel to rArtikelbeheer;
grant all on Artikelgroep to rArtikelbeheer;
grant select on Orderregel to rArtikelbeheer;
grant select on Klacht to rArtikelbeheer;

create role rVerkoop;
grant all on Order_ to rVerkoop;
grant all on Orderregel to rVerkoop;
grant select on Artikel to rVerkoop;
grant select on Artikelgroep to rVerkoop;
grant select on Kortingsinterval to rVerkoop;
grant select, insert on vKlant to rVerkoop;
grant references on Klant to rVerkoop;
```

12.18 SQL-statement:

```
grant select on Artikel to public
```

12.19 De gebruikers laten we aanmaken door Sysdba. Het maakt daarbij niet uit op welke database we Sysdba laten 'connecten', als het maar een bestaande database is:

```
connect 'OrderdatabaseC.fdb' user 'Sysdba' password 'masterkey';
create user Pablo password 'pw';
create user Pepe password 'pw';
create user Rosa password 'pw'
```

Hierna loggen we weer in als gebruiker OrderdatabaseC, waarna we de rollen en privileges kunnen toekennen:

```
connect 'OrderdatabaseC.fdb' user 'OrderdatabaseC' password 'pw';
grant rVerkoop to Pablo, Rosa;
grant rArtikelbeheer to Pepe, Rosa;
grant all on Klacht to Rosa with grant option
```

Een alternatief voor de connect-commando's is: inloggen via het databasemenu van de Boekverkenner.

12.20a Het commando geeft geen foutmelding, maar het effect is nihil: Pablo heeft dit leesrecht niet rechtstreeks ontvangen als gebruiker, en wat hem niet is toegekend, kan hem ook niet worden afgenomen. Via twee wegen kan Pablo het leesrecht op Artikel blijven uitoefenen: via public en via rVerkoop. Zolang public dit recht heeft, kan Pablo het uitoefenen ongeacht of hij via de rol vVerkoop heeft ingelogd. Zou public het verliezen, dan kan Pablo het nog slechts uitoefenen door in te loggen met de rol vVerkoop.

b Het effect is dat de rol rVerkoop het delete-privilege op Order_ kwijtraakt. Een gebruiker aan wie deze rol is toegekend en die vervolgens via deze rol inlogt, kan dus geen orderrijen meer verwijderen, tenzij het delete-privilege ook rechtstreeks is verkregen.
De overige rechten op Order_, via grant all ... verkregen, blijven gehandhaafd. all is immers geen zelfstandig privilege, maar de bundeling van select, insert, delete, update en references. In de data dictionary zal men all dan ook niet terugvinden, maar wel de genoemde onderliggende privileges.

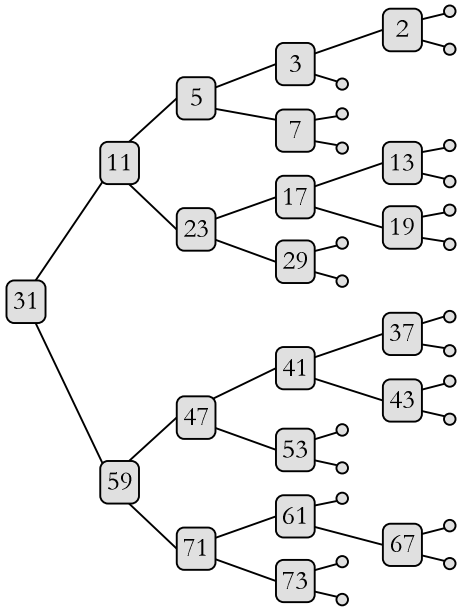
12.21 SQL-statements:

```
connect 'OrderdatabaseC.fdb'
   user 'Rosa' password 'pw' role 'rVerkoop';
insert into vKlant values (1500, 'Stern');
insert into Order_ (nr, klant) values (5900, 1500);
insert into Orderregel values (5900, 1, 238, 10, 150.10);
insert into Orderregel values (5900, 2, 107, 100, 198);
```

12.22 Rollen en domeinen zijn beide voortgekomen uit de behoefte een 'single point of definition' te creëren. Bij domeinen gaat het om datatypen en eventueel ook constraints op kolommen, die niet toevallig maar vanwege hun betekenis overeenkomen. Bij rollen gaat het om verzamelingen van privileges van verschillende gebruikers, die niet toevallig maar vanwege hun werkzaamheden met betrekking tot bepaalde bedrijfsprocessen, overeenkomen.

Uitwerkingen hoofdstuk 13

13.1 Er zijn meerdere goede oplossingen. Dit is er een (zie figuur 13.11).



Figuur 13.11 Mogelijke oplossing indexboom

13.2 Figuur 13.12 geeft de resultaten op ons systeem grafisch weer.

	10	50	100	500	1000	5000	10000
zonder	2,42	2,4	2,34	2,28	2,26	2,25	2,25
met	2,69	2,42	1,92	1	0,64	0,18	0

Figuur 13.12 Tabel met grafiek

Zonder index wordt de gehele tabel Orderregel doorlopen, wordt van ieder record nagegaan of bedrag groter is dan *n* en zo ja dan wordt het totaal met 1 opgehoogd.
Het doorlopen van de tabel en het testen of bedrag aan de gestelde voorwaarde voldoet is onafhankelijk van de waarde van *n*. Omdat er bij grote *n* minder records voldoen, zal het bijwerken van het totaal wat sneller gaan. Dit verklaart het langzaam afnemen van de benodigde tijd, zoals te zien in de gegevens van reeks 1.

Bij reeks 2 worden dankzij de index snel de records gevonden die voldoen aan de voorwaarde. Maar daarna moeten de gevonden records sequentieel worden doorlopen om het aantal te bepalen. Dit kost iets meer tijd dan het doorlopen van de tabel zonder index omdat bij iedere waarde van de index het bijbehorende record gezocht moet worden. Bij bedrag > 10 kost dit iets meer tijd dan in het geval zonder index: dit punt van de grafiek van reeks 2 ligt boven dat van reeks 1. Voor hogere waarden van *n* is de versnelling door het gebruik van de index iBedrag zeer goed merkbaar.

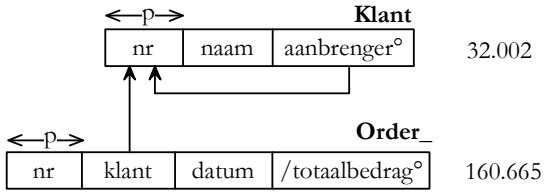
13.3 We krijgen :

```
select *
from   Order_ O
       join Klant K on O.klant = K.nr
```

Het plan hiervoor luidt :

```
PLAN JOIN (K NATURAL, O INDEX (FK_ORDER_VAN_KLANT))
```

Figuur 13.13 toont de betrokken tabellen, met hun aantallen records.



Figuur 13.13

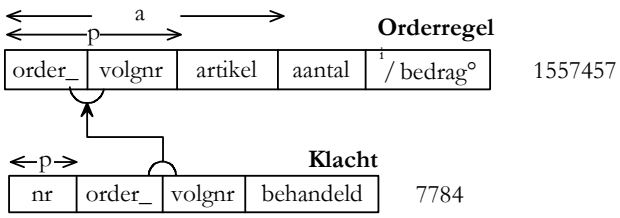
De andere query luidt :

```
select  *
from    Orderregel Orl
        join Klacht Kt on Orl.order_ = Kt.order_
                          and Orl.volgnr = Kt.volgnr
```

met als plan :

```
PLAN JOIN (KT NATURAL, ORL INDEX (PK_ORDERREGEL))
```

Figuur 13.14 toont de betrokken tabellen, met hun aantallen records.



Figuur 13.14

Het patroon van beide queryplannen is gelijk: er wordt gestart bij de kleinste tabel en via een index op de grootste tabel worden daar de corresponderende records bijgezocht. Dit geeft een verschil in de gebruikte index:
- Bij de eerste query wordt gestart in Klant (oudertabel). Daar worden de corresponderende records van Orderregel (kindtabel) bijgezocht via de index op de verwijssleutel van Orderregel.
- Bij de tweede query wordt gestart in Klacht (kindtabel) en worden daar de corresponderende records van Orderregel (oudertabel) bijgezocht via de index op de primaire sleutel van Orderregel.

13.4 De query:

```
select   first 10 *
from     Orderregel
order by order_ desc, volgnr desc
```

Op ons systeem was de benodigde tijd 18.92 s. Deze lange tijd wordt veroorzaakt doordat het gehele bestand eerst gesorteerd moet worden. Het gaat sneller als we een index maken op basis van de sorteervolgorde:

```
create desc index iNr2 on Orderregel(order_, volgnr)
```

Voor de sortering wordt index iNr2 gebruikt, wat resulteert in een zeer snelle uitvoering van de query: 0.01 s.

13.5 We krijgen achtereenvolgens:

```
--a
select count(*)
from   Orderregel Orl
where  Orl.order_|| '/' || Orl.volgnr
       not in (select Kt.order_|| '/' || Kt.volgnr
               from   Klacht Kt)
```

Deze query duurt veel te lang; we moeten de uitvoering afbreken. Helaas kan dat alleen hardhandig: het proces stoppen via het besturingssysteem.

```
--b
select count(*)
from   Orderregel Orl
where  not exists (select *
                   from   Klacht Kt
                   where  Orl.order_ = Kt.order_
                          and Orl.volgnr = Kt.volgnr)
```

Op ons systeem duurde deze query 9.70 s.

Het alternatief met een left outer join:

```
--c
select count(*)
from   Orderregel Orl
       left outer join Klacht Kt on Orl.order_ = Kt.order_
                                    and Orl.volgnr = Kt.volgnr
where  behandeld is null
```

Deze query nam 10.00 s in beslag.

Bij de a-query kan geen index gebruikt worden. Omdat voor ieder record van zowel tabel Orderregel alsook voor die van Klacht een functie moet worden uitgevoerd, duurt de verwerking erg lang.

13.6a en b Op ons systeem maten we de volgende resultaten.

	verwerkingstijd	*queryplan*
q1 zonder index	0.08	Klant natural
q2 zonder index	0.11	Klant natural
q1 met index	0.00	Klant index(iNaam)
q2 met index	0.11	Klant natural

Als we geen index gebruiken is de verwerkingstijd voor q2 hoger dan voor q1 ten gevolge van het gebruik van de functie substring.
Als we een index hebben gemaakt, wordt deze bij q1 wel gebruikt en bij q2 niet. Gevolg is dat q1 veel sneller gaat maar q2 evenveel tijd kost. In dit geval levert het formuleren van een query zonder gebruik van een functie winst op, vooral als ook een extra index gemaakt wordt.

13.7 We krijgen nu:

```
select  count(*)
from    Orderregel
where   nr not in (select orderregel
                   from   Klacht)
```

De originele query duurde uren, deze versie 13.2 s.

```
select   count(*)
from     Orderregel Orl
where    not exists (select *
                     from   Klacht
                     where  orderregel = Orl.nr)
```

Benodigde tijd 9.80s, weinig verschil met originele query.

De laatste query met de left outer join wordt nu:

```
select count(*)
from   Orderregel Orl
       left outer join Klacht K on Orl.nr = K.orderregel
where  behandeld is null
```

Ook hier is de benodigde tijd nauwelijks verschillend van die van de originele query, zo rond de 10.0 s.

13.8a Aantallen rijen:
- start met Artikel: 200
- left outer join met Artikelgroep: 200
- join met orderregel: 1.500.000
- join met Klacht: 7500
- pas selectieconditie toe: 400

Het maximum aantal rijen is dus 1.500.000.

b De volgende volgorde vergt aanzienlijk minder geheugen:
- start met Klacht: 7500
- pas selectieconditie toe: 400
- join met Orderregel: 400
- join met Artikel: 400
- join met Artikelgroep: 400

Het maximum aantal rijen is nu 7500. In werkelijkheid kunnen deze aantallen wel wat verschillen, daar de rijen vanaf schijf per 'page' (een fysieke lees-/schrijfeenheid) worden gelezen. Bovendien hebben we alleen naar de aantallen rijen gekeken, terwijl de grootte van een rij (in bytes) ook een rol speelt.

```
PLAN SORT( JOIN
           (JOIN
             (JOIN (A NATURAL, AG INDEX (PK_ARTIKELGROEP)),
              ORL INDEX (FK_ORDERREGEL_BIJ_ARTIKEL)
             ),
             KHT INDEX (FK_KLACHT_BIJ_ORDER_REGEL)
           )
         )
```

13.9 Volgens het queryplan worden er drie joins gemaakt:
- van Artikel met Artikelgroep
- het resultaat wordt gejoind met Orderregel
- dit resultaat wordt weer gejoind met Klacht

Het resultaat van de joins worden gesorteerd (SORT);

```
      NR ARTIKEL      ARTIKELGROEP     COUNT
======== ============ ============ ========
       2 Bakwiel      niznomoeren          2
       3 Bandklem     niznomoeren          3
       4 Belbak       <NULL>               4
       5 Belbek       <NULL>               2
       6 Berkel       <NULL>               1
       7 Blaffel      niznomoeren          2
       8 Bradel       voegringen           1
       9 Bradelbak    voegringen           5
       … …            …                    …
     201 Zwinzwalik   voegringen           2
     202 Zwipstaart   voegringen           2
```

13.10 De conditie op de Artikel-tabel resulteert in slechts één rij. Dit maakt het voordelig daar het fysieke joinnavigatiepad te starten. Dit leidt tot het volgende fysieke pad en queryplan.

Artikel (en daarin zoeken van artikel 152 via de primaire-sleutelindex)
→ Artikelgroep (gebruikmakend de index op de primaire sleutel)
→ Orderregel (gebruikmakend de index op de verwijssleutel)
→ Klacht (gebruikmakend de index op de verwijssleutel).

Dit komt overeen met het door Firebird gekozen queryplan:

```
PLAN SORT (JOIN (A INDEX (RDB$PRIMARY8),AG INDEX (RDB$PRIMARY6),ORL INDEX (RDB$FOREIGN14),KHT INDEX (RDB$FOREIGN17)))
```

Merk op dat twee keer wordt gejoind in de richting ouder-kind, waarbij de verwijssleutelindex van de kindtabel wordt gebruikt om bij een ouderrij de corresponderende kindrijen te zoeken.

13.11 Zonder index wordt het queryplan:

```
PLAN SORT ((ARTIKEL NATURAL))
```

We krijgen nu een 'natural sort', dat wil zeggen: de resultaattabel wordt geordend zonder hulp van een index. Na aanmaken van een index:

```
create index iArtikelverkoopprijs on Artikel(verkoopprijs)
```

wordt voor het ordenen de index gebruikt:

```
PLAN (ARTIKEL ORDER IARTIKELVERKOOPPRIJS)
```

13.12 De tabel Artikelgroep is klein en zal dat altijd blijven. Zoeken en joinen met deze tabel zal daarom altijd snel gaan. Het argument dat zoeken en joinen bij numerieke sleutels doorgaans sneller gaat dan bij alfanumerieke, is daarom voor Artikelgroep niet van toepassing. De keuze voor codes is nu voordelig, mits deze betekenisvol zijn. Immers: bij betekenisvolle codes (bijvoorbeeld 'NM') zal het minder vaak nodig zijn om ook de artikelgroepomschrijving ('niznomoeren') op te halen (wat weer een join vraagt).

Uitwerkingen hoofdstuk 14

14.1 Insert-script:

```
insert into Reis values (38, '03-mar-2021', 'RV', 520, 3.35);
insert into Bezoek values (38, 1, 'Venus', 3);
insert into Bezoek values (38, 2, 'Aarde', 0);
insert into Bezoek values (38, 3, 'Mars',  7);
insert into Klant values (127, 'Linda', '21-jun-1985');
insert into Deelname values (38, 127)
```

(Als u dit script uitvoert, geef dan na afloop een rollback.)

14.2 Er worden reizen gevraagd, dus we navigeren vanuit tabel Reis, en passen de volgorden daaraan aan, ongeacht de oplossing die we kiezen. Eerst geven we een oplossing met een join:

```
select distinct R.nr, R.vertrekdatum, R.duur
from   Reis R
       join Bezoek B on R.nr = B.reis
where  B.hemelobject = 'Mars'
       and B.verblijfsduur > 0   -- landing
```

De `distinct` is nodig omdat een reis in principe meer dan één verblijf op Mars kan bevatten. In plaats van pseudo-groeperen met `distinct` kun je ook echt groeperen op de drie kolommen van de `select`-lijst. Maar een subselect is hier misschien toch het mooist:

```
select nr, vertrekdatum, duur
from   Reis
where  nr in (select reis
              from   Bezoek
              where  hemelobject = 'Mars'
                     and verblijfsduur > 0)
```

14.3 Om geen hemelobjecten kwijt te raken gebruiken we een left outer join:

```
select H.naam, H.moederobject, M.moederobject
from   Hemelobject H
       left outer join Hemelobject M on H.moederobject = M.naam
```

Met M.naam in plaats van H.moederobject in de `select`-lijst krijg je natuurlijk ook een goed resultaat. We vinden dit minder mooi, want alleen vanwege de derde kolom wordt gejoind met de M-alias van Hemelobject (zeg maar het 'M-exemplaar'). Als principe zou je kunnen formuleren: steek – al navigerend door de database – zo laat mogelijk over naar een volgende tabel.

14.4 We groeperen om de aantallen te krijgen en met `distinct` onderdrukken we doublures:

```
select distinct count(*)
from   Hemelobject
where  moederobject <> 'Zon'
       and moederobject is not null
group by moederobject
```

14.5 In uitwerking 14.4 werd `distinct` toegepast op het resultaat van de telling. Hier is een `distinct` nodig die voorafgaat aan de telling:

```
select   reis, count(distinct hemelobject)
from     Bezoek
group by reis
```

14.6a Oplossing met nesting van statistische functies (geen Firebird):

```
select   R.reis, R.vertrekdatum
from     Reis R
         join Deelname D on R.nr = D.reis
group by R.nr, R.vertrekdatum
having   count(*) = (select max(count(*))
                     from   Deelname
                     group by reis)
```

b Oplossing met viewdefinitie gevolgd door `select`-statement:

```
create view vReisPlus (nr, vertrekdatum, duur, prijs, aantal_deelnemers)
as select   R.nr, R.vertrekdatum, R.duur, R.prijs, count(*)
   from     Reis R
            join Deelname D on R.nr = D.reis
   group by R.nr, R.vertrekdatum, R.duur, R.prijs;

select nr, vertrekdatum, aantal_deelnemers
from   vReisPlus
where  aantal_deelnemers = (select max(aantal_deelnemers)
                            from   vReisPlus)
```

De viewdefinitie is uiteraard eenmalig. Hij bevat meer kolommen dan strict nodig, met het oog op algemener gebruik.

14.7 De voorwaarde vertalen we rechttoe-rechtaan in een subselect:

```
select nr
from   Reis R
where  1 = (select count(distinct hemelobject)
            from   Bezoek
            where  reis = R.nr)
```

Tellen van de '`distinct` hemelobject' is nodig omdat tijdens een reis een hemelobject meerdere keren kan worden bezocht. Als alternatief voor de gecorreleerde subselect kun je ook joinen en groeperen:

```
select   R.nr
from     Reis R
         join Bezoek B on R.nr = B.reis
group by R.nr
having   count(distinct B.hemelobject) = 1
```

14.8 Ook hier hanteren we een stapsgewijze aanpak met een subselect:

```
select nr, vertrekdatum, duur
from   Reis
where  nr in (-- reisnummers van bezoeken aan Mars
              select reis
              from   Bezoek B
                     join Hemelobject H on B.hemelobject = H.naam
              where  H.moederobject = 'Mars')
```

Alternatief: gecorreleerde subselect met `exists`.

14.9 Wederom een stapsgewijze aanpak, waarbij de `where`-conditie uitdrukt: 'er is tijdens deze reis een landing op een planeet':

```
select R.nr
from   Reis R
where  exists
       (-- bijbehorende Bezoek-rijen met landing op planeet
        select *
        from   Bezoek B
               join Hemelobject H on B.hemelobject = H.naam
        where  B.reis = R.nr                -- bijbehorende Bezoek-rij
               and B.verblijfsduur > 0      -- landing
               and H.moederobject = 'Zon')  -- planeet
```

14.10a Een enkele subselect volstaat:

```
select nr, vertrekdatum
from   Reis
where  nr in (select reis
              from   Bezoek
              where  (hemelobject = 'Mars') or (hemelobject = 'Maan'))
```

De conditie binnen de subselect kan ook geformuleerd worden als: `naam in ('Mars', 'Maan')`, een vorm die zeker de voorkeur verdient bij drie of meer hemelobjecten.

b De Mars-conditie en de Maan-conditie hebben een eigen subselect nodig:

```
select nr, vertrekdatum
from   Reis
where  nr in (select reis
              from   Bezoek
              where  hemelobject = 'Mars')
       and
       nr in (select reis
              from   Bezoek
              where  hemelobject = 'Maan')
```

14.11 Nee, dit leidt tot de foutmelding dat kolom 'totale_verblijfsduur' onbekend is. Dat is ook te voorspellen vanuit de conceptuele verwerkingsvolgorde: de `select`-clausule wordt als laatste verwerkt, dus vóór die tijd is de kolomalias totale_verblijfsduur nog niet bekend.

14.12 In deze uitwerking illustreren we hoe ook een overhaaste aanpak bij stap 1 tot een onbevredigende oplossing kan leiden.

Stap 1: is de vraagstelling 100% duidelijk?
Er wordt kennelijk gevraagd naar verboden boekingen: boekingen die in strijd zijn met een deelname aan een andere reis. Wat zijn echter boekingen? We moeten weer even te rade gaan bij onze denkbeeldige deskundige en nemen aan dat deze bevestigt wat voor de hand ligt: dat een boeking overeenkomt met een deelname.

Stap 2: hoe zou je het probleem handmatig oplossen?
Door de deelnames op volgorde van vertrekdatum af te werken en bij elke deelname te kijken of er een al gecontroleerde deelname bestaat waarmee 'deze' deelname in strijd is.

Stap 3: kan het ook anders, handiger misschien?
Dat lijkt niet aannemelijk, de geschetste aanpak is rechttoe-rechtaan.

Stap 4: stapsgewijze transformatie naar SQL
Vraag en antwoord (zie paragraaf 14.2 voor de genummerde vragen):

Vraag 1:	Waarover wordt informatie gevraagd?
Antwoord:	Over verboden deelnamen. Het navigatiepad start dus in Deelname.
Vraag 2:	In welke tabellen staan de gevraagde gegevens?
Antwoord:	In Deelname, Reis en Klant.

Dit leidt tot:

```
select D.reis, R.vertrekdatum, D.klant, K.naam
from   Deelname D
       join Reis R on D.reis  = R.nr
       join Klant K on D.klant = K.nr
where  dit is een verboden deelname
```

De 'rechttoe-rechtaan'-aanpak (zie stap 3) leidt via de volgende vraag/antwoord-combinatie:

Vraag:	Welke zijn die verboden deelnamen?
Antwoord:	Die waarbij een ándere deelname van dezelfde klant bestaat, met een vertrekdatum die eerder is dan of gelijk is aan die van deze (verboden) deelname, en waarvan de terugkeerdatum-plus-30-dagen nog niet is verstreken op de vertrekdatum van deze deelname.

tot:

```
select D.reis, R.vertrekdatum, D.klant, K.naam
from   Deelname D
       join Reis R on D.reis  = R.nr
       join Klant K on D.klant = K.nr
where  -- dit is een verboden deelname
       exists
       (een andere deelname van dezelfde klant, met een gelijke of vroegere vertrekdatum  dan die van deze (verboden)
       deelname, en waarvan de terugkeerdatum-plus-30-dagen nog niet is verstreken op de vertrekdatum van deze deelname)
```

Wat na `exists` in natuurlijke taal is geformuleerd, is een verzameling, de verzameling van alle deelnamen die aan de beschreven conditie voldoen. In de volgende stap wordt deze uitgewerkt als subselect:

```
select D.reis, R.vertrekdatum, D.klant, K.naam
from   Deelname D
       join Reis R on D.reis  = R.nr
       join Klant K on D.klant = K.nr
where  -- dit is een verboden deelname
       (-- een andere deelname van dezelfde klant, met een gelijke of vroegere vertrekdatum dan die van deze
        -- (verboden) deelname,en waarvan de terugkeerdatum-plus-30-dagen nog niet is verstreken op de vertrekdatum
        -- van deze deelname
deelname
        select *
        from   Deelname D1
               join Reis R1 on D1.reisnr = R1.reisnr
        where  dit is een andere deelname van dezelfde klant
               and vertrekdatum van 'andere' deelname is zelfde of eerdere  datum dan vertrekdatum van 'deze' deelname
               and terugkeerdatum-plus-30-dagen van 'andere' deelname is nog niet verstreken op vertrekdatum van deze
deelname)
```

De eindoplossing wordt nu:

```
select D.reis, R.vertrekdatum, D.klant, K.naam
from   Deelname D
       join Reis R on D.reis  = R.nr
       join Klant K on D.klant = K.nr
where  -- dit is een verboden deelname
       exists
       (-- een andere deelname van dezelfde klant, met een gelijke of vroegere vertrekdatum dan die van deze
        -- (verboden) deelname,en waarvan de terugkeerdatum-plus-30-dagen nog niet is verstreken op de vertrekdatum
        -- van deze deelname
        select *
        from   Deelname D1
               join Reis R1 on D1.reis = R1.nr
        where  -- dit is een andere deelname van dezelfde klant
               D1.reis <> D.reis
               and D1.klant = D.klant
               and -- vertrekdatum van 'andere' deelname is zelfde of eerdere datum dan vertrekdatum van 'deze'
                   -- deelname
                   R1.vertrekdatum <= R.vertrekdatum
               and -- terugkeerdatum-plus-30-dagen van 'andere' deelname is nog niet verstreken op vertrekdatum
                   -- van 'deze' deelname
                   R1.vertrekdatum + R1.duur + 30 >= R.vertrekdatum)
```

Er blijkt één overtreding te zijn:

```
    REIS VERTREKDATUM     KLANT NAAM
======== ============ ========= ======
      33 12-okt-2022        121 Arne
```

Kritische beschouwing van deze aanpak
Nadere inspectie leert dat klant Arne op de vertrekdatum van reis 33 nog niet terug is van reis 32. Jammer is nu toch wel dat we dit nog apart op moeten zoeken. Verder kunnen we ons afvragen welke van beide deelnamen van klant 121 de eigenlijke overtreding is. Er is géén boekingsdatum, dus misschien heeft de klant reis 32 wel eerder geboekt dan reis 33! In dat geval zou de boeking voor reis 32 de overtreding zijn. Terugkerend naar stap 1 ('Is de vraagstelling 100% duidelijk?') kunnen we concluderen dat het correcter is om de combinatie van beide deelnamen als de overtreding te zien

en dus naar combinaties te vragen. De volgende oplossing is daarop gebaseerd. Omdat het gevraagde overzicht nu uitgebreider is, worden we vanzelf tot een 'bredere' join gedwongen, *een mogelijkheid die we bij de hiervóór gegeven oplossing óók hadden kunnen kiezen maar daar niet hebben overwogen*! Omdat de bredere join de subselect overbodig maakt, geven we deze oplossing in één keer. Het taalgebruik passen we enigszins aan: in plaats van 'andere' en 'deze' wordt het: 'eerdere deelname' (alias D1) en 'latere deelname' (alias D2), met corresponderende aliassen (R1 en R2) voor de reizen:

```
select D1.klant,
       K.naam,
       D1.reis                          reisnr1,
       R1.vertrekdatum                  vertrekdatum1,
       R1.vertrekdatum + R1.duur        geplande_aankomstdatum1,
       R2.nr                            reisnr2,
       R2.vertrekdatum                  vertrekdatum2
from   -- D1: 'eerdere deelname' D2: 'latere deelname'
       -- (betreft vertrekdata, die eventueel gelijk mogen zijn)
       Deelname D1
       join Deelname D2 on D1.klant = D2.klant     -- zelfde klant
                       and D1.reis <> D2.reis  -- andere reis
       join Reis R1 on D1.reis  = R1.nr
       join Klant K on D1.klant = K.nr
       join Reis R2 on D2.reis  = R2.nr
where  -- dit is een verboden combinatie, want:
       -- 1. vertrekdatum van D1-deelname is zelfde of eerdere datum dan vertrekdatum van D2-deelname
       R1.vertrekdatum <= R2.vertrekdatum
       and
       -- 2. terugkeerdatum-plus-30-dagen van D2-deelname is nog niet verstreken op vertrekdatum van D1-deelname
       R1.vertrekdatum + R1.duur + 30 >= R2.vertrekdatum
```

Resultaat:

```
    KLANT NAAM     REISNR1 VERTREKDATUM1 GEPLANDE_AANKOMSTDATUM1   REISNR2 VERTREKDATUM2
======== ====== ======== ============ ====================== ======== ============
      121 Arne          32 03-jun-2022   28-jun-2023                  33 12-okt-2022
```

Terugkijkend kunnen we vaststellen dat we stap 1 aanvankelijk slordig hadden afgehandeld, en dat we bij bij het vergelijken van 'deze deelname' met andere deelnamen direct voor subselectnavigatie hebben gekozen zonder joinnavigatie te overwegen.
Een relativering is hier wel op zijn plaats: in de praktijk doe je het zelden in één keer optimaal, welke fasering je ook hanteert. Het enige dat écht telt, is kritisch te blijven kijken naar alles wat je hebt gemaakt.

14.13 We schrijven *aantal deelnemers* gewoon uit:

```
select nr, (-- aantal deelnemers aan 'deze' reis
            select count(*)
            from   Deelname
            where  reis = R.nr)   aantal_deelnemers
from   Reis R
```

Toelichting:
- Het gevraagde aantal deelnemers wordt berekend via een gecorreleerde subselect. Deze levert voor elke reis (de actuele rij van de hoofdselect) precies één waarde voor de resultaatkolom, die als kolomnaam (alias) aantal_deelnemers heeft.
- 'Reis R' (achter `from`) gaat conceptueel vooraf aan de `select`-clausule, zodat het 'deze' in de eerste regel en de alias `R` correcte *terug*verwijzingen zijn.

De tweede oplossing luidt:

```
select   R.nr, count(D.reis)
from     Reis R
         left outer join Deelname D on R.nr = D.reis
group by R.nr
```

14.14 De vraagstelling is voor meer dan één uitleg vatbaar. We kiezen de volgende precisering: geef reisnummer en vertrekdatum van de ruimtereis (of ruimtereizen, het kunnen er meerdere zijn) waarvoor het aantal door één of meer deelnemers tijdens eerdere reizen bezochte hemelobjecten het grootst is.
De probleem is dermate complex dat we eerst een view definiëren met daarin de aantallen waarvan we later het maximum nodig hebben:

```
create view vReisMetAantalEerderBezocht
            (nr, vertrekdatum, aantal_eerder_bezocht)
as
select DezeReis.nr, DezeReis.vertrekdatum, count(distinct hemelobject)
from   Reis DezeReis
       join Deelname D_DezeReis on DezeReis.nr = D_DezeReis.reis
       join Deelname D_EerdereReis
            on D_DezeReis.klant = D_EerdereReis.klant
       join Reis EerdereReis
            on D_EerdereReis.reis = EerdereReis.nr
               and EerdereReis.vertrekdatum < DezeReis.vertrekdatum
       join Bezoek B_EerdereReis
            on EerdereReis.nr = B_EerdereReis.reis
group by DezeReis.nr, DezeReis.vertrekdatum
```

Merk op dat we ervoor hebben gekozen de 'eerdere reis'-voorwaarde in een joinconditie op te nemen. Hierna is het snel geklaard:

```
select nr, vertrekdatum
from   vReisMetAantalEerderBezocht
where  aantal_eerder_bezocht = (select max(aantal_eerder_bezocht)
                                from   vReisMetAantalEerderBezocht)
```

Resultaat:

```
      NR VERTREKDATUM
======== ===========
      33 12-okt-2022
```

14.15 Merk in de volgende oplossing op dat de joinconditie garandeert dat we met een maan van een planeet van doen hebben:

```
select   Maan.naam, Maan.moederobject
from     Hemelobject Maan
         join Hemelobject Planeet
              on Maan.moederobject = Planeet.naam
                 and Planeet.moederobject = 'Zon'
where    Maan.afstand = (select max(afstand)
                         from   Hemelobject
                         where  moederobject = Maan.moederobject)
```

14.16 Zoals veel problemen met 'alle' lossen we ook dit probleem op met een geneste (`not`) `exists`-constructie:

```
select nr, vertrekdatum
from   Reis R
where  nr <> 35
       and not exists
           (-- hemelobjecten die tijdens deze reis niet worden bezocht en die wel worden bezocht tijdens reis 35
            select *
            from   Hemelobject H
            where  not exists
                   (-- een bezoek tijdens deze reis aan dit hemelobject
                    select *
                    from   Bezoek
                    where  reis = R.nr
                           and hemelobject = H.naam)
                   and exists
                   (-- een bezoek tijdens reis 35 aan dit hemelobject
                    select *
                    from   Bezoek
                    where  reis = 35
                           and hemelobject = H.naam))
```

14.17 Eerste stap:

```
select nr, naam
from   klant K
where  nr in (klantnummers van klanten die aan reis 33 deelnamen)
       and
       not exists (een eerdere reis van 'deze' klant)
```

Uitwerken van de condities tot subselects geeft:

```
select nr, naam
from   klant K
where  nr in
           (-- klantnummers van klanten die aan reis 33 deelnamen
            select klant
            from   Deelname
            where  reis = 33)
       and not exists
           (-- een eerdere reis van deze klant
            select *
            from   Deelname D
                   join Reis R on D.reis = R.nr
            where  klant = K.nr
                   and vertrekdatum < (select vertrekdatum
                                       from   Reis
                                       where  nr = 33))
```

Alternatief:

```
select K.nr, K.naam
from   Klant K
       join Deelname D on K.nr = D.klant
       join Reis Huidige on D.reis = Huidige.nr
where  Huidige.nr = 33
       and not exists
       (-- vorige reizen van dezelfde klant
        select *
        from   Deelname D1
               join Reis Vorige on D1.reis = Vorige.nr
        where  -- dezelfde klant, vorige reis
               D1.klant = K.nr
               and Vorige.vertrekdatum < Huidige.vertrekdatum)
```

14.18 Bijna een standaardprobleem uit de categorie 'minimax'! Een kleine complicatie is dat meervoudig bezochte hemelobjecten slechts enkelvoudig mogen worden geteld. Maar dat lossen we eenvoudig op met een `distinct`:

```
select nr, vertrekdatum
from   Reis
where  nr in
       (select   reis
        from     Bezoek
        group by reis
        having   count(distinct hemelobject) =
                 (-- grootste aantal verschillende hemelobjecten
                  -- tijdens één reis
                  select   max(count(distinct hemelobject))
                  from     Bezoek
                  group by reis))
```

In Firebird kunnen we statistische functies niet nesten. We laten nog eens zien hoe eenvoudig dat vermeden kan worden (zie markering):

```
select nr, vertrekdatum
from   Reis
where  nr in
       (select   reis
        from     Bezoek
        group by reis
        having   count(distinct hemelobject) =
                 (-- grootste aantal verschillende hemelobjecten
                  -- tijdens één reis
                  select   max(aantal)
                  from     (select   count(distinct hemelobject) aantal
                            from     Bezoek
                            group by reis)))
```

14.19a Het probleem lijkt een standaard ouders-met-aantal-kinderen-probleem, zoals in voorbeeld 14.15 het deelprobleem van de planeten met hun aantal manen. Overnemen van de daar toegepaste oplossingswijze leidt tot:

```
select   H.naam, count(B.hemelobject)
from     Hemelobject H
         left outer join Bezoek B on H.naam = B.hemelobject
where    B.verblijfsduur > 0
group by H.naam
order by H.naam
```

Echter, het resultaat is niet wat we willen:

```
NAAM       COUNT
====== =========
Io             2
Maan           4
Mars           5
Phobos         1
```

Dat komt doordat de left outer join weliswaar nog álle hemelobjecten meeneemt, maar de selectieconditie in de `where`-clausule daarna alsnog alle joinrijen laat sneuvelen waarvan de Bezoek-component géén landing betreft. En daarmee raken we dus alle hemelobjecten zonder landing kwijt.
Iets algemener gesteld: ondanks een left outer join raken we ongewild rijen kwijt vanwege een selectieconditie. De oplossing is: neem de selectieconditie op in de left-outer-joinconditie:

```
select   H.naam, count(B.hemelobject)
from     Hemelobject H
         left outer join Bezoek B on H.naam = B.hemelobject
                                     and B.verblijfsduur > 0
group by H.naam
order by H.naam
```

De operator `left outer join` werkt hier zoals altijd: eerst een inner join toepassen met de gegeven `on`-conditie (`H.naam = B.hemelobject and B.verblijfsduur > 0`), en vervolgens alle rijen die hiermee zouden verdwijnen toch opnemen.

Resultaat:

```
NAAM     COUNT
==== =========
Aarde        0
...         ..
Io           2
...         ..
Maan         4
Mars         4
...         ..
Phobos       1
...         ..
Zon          0
```

b De tweede oplossing is rechttoe-rechtaan. Vanuit een eerste stap:

```
select   naam, aantal landingen op 'dit' hemelobject
from     Hemelobject
order by naam
```

volgt, zonder complicaties, als oplossing:

```
select   naam, (select count(*)
                from   Bezoek
                where  hemelobject = H.naam
                       and verblijfsduur > 0) aantal_landingen
from     Hemelobject H
order by naam
```

Uitwerkingen hoofdstuk 15

15.1 Geen uitwerking.

15.2 Het is inderdaad heel gewoon dat gegevens die de ene gebruiker heeft ingevoerd, op een gegeven moment worden overschreven door iemand anders. Het probleem zit hem in het feit dat in het voorbeeld de eerste medewerker in de veronderstelling is dat hij met een *transactie* bezig is: een aantal bij elkaar horende databaseacties, die óf allemaal wel óf allemaal niet worden uitgevoerd. Anders gezegd: hij gaat ervan uit dat zijn acties één geheel vormen, waar niets tussen kan komen zolang hij daarmee bezig is. (In ACID-termen: een transactie moet atomair en geïsoleerd zijn.) Hij mag van het rdbms verwachten dat dat ervoor zorgt dat zijn transactie op die manier wordt uitgevoerd. Als er een lost update optreedt, heeft het rdbms dat duidelijk niet goed gedaan.

Opmerking: dat laatste is niet noodzakelijk een fout van het rdbms; in paragraaf 2 zien we hoe een gebruiker per transactie kan aangeven hoe strikt het rdbms erop moet letten dat andere transacties hem niet in de wielen rijden.

15.3a Transactie A doet hier een dirty read: onder de gegevens die uit tabel Reis worden gelezen bevindt zich een ongecommitte rij die door transactie B is toegevoegd.

b Dit is een voorbeeld van een lost update van transactie B. Het feit dat slechts een deel van B's update verloren gaat doet er niet toe.

Opmerking: u kunt dit scenario later simuleren; u zult zien dat u dan een 'lock conflict' krijgt, net als in de uitwerking van opgave 15.6.

c Transactie B leest tweemaal dezelfde rij in tabel Bezoek, maar zal de tweede keer een ander resultaat krijgen omdat transactie A intussen die rij heeft verwijderd: een standaardvoorbeeld van een non-repeatable read door B. In voorbeeld 15.8 simuleren we dit scenario in IQU.

d De update van transactie A gaat verloren door de delete van transactie B; een klassiek geval van een lost update dus.

Opmerking: Als u wilt kunt u deze opgave later uitproberen en constateren dat er inderdaad een 'lock conflict' optreedt.

e Dit is een strikvraag: de delete van B gaat niet door vanwege de restricted delete op de verwijzing van Reis naar Transport. In termen van concurrency gaat er dus niets mis, want B krijgt een foutmelding en de delete wordt niet uitgevoerd.

15.4 Dit betekent dat lost updates niet voor kunnen komen, welk isolation level je ook kiest.

15.5 Met isolation level read committed zie je alleen gecommitte wijzigingen van andere transacties. Zolang transactie A niet gecommit heeft, zou transactie B dus geen wijziging moeten zien. Simulatie van dit scenario met IQU bevestigt dit.

15.6a Dit is een voorbeeld van een update-conflict (zie ook de opmerking bij de bespreking van het isolation level serializable uit de SQL-standaard). Vanwege de eis dat lost updates nooit – bij geen enkel isolation level – mogen voorkomen heeft Firebird zichzelf de volgende beperking opgelegd (zie ook opgave 15.4):

"When simultaneous transactions attempt to update the same data in tables, only the first update succeeds. No other transaction can update or delete that data until the controlling transaction is rolled back or committed".

Het maakt dan ook niet uit welk isolation level we kiezen: in alle gevallen krijgen we een update-conflict, en een foutmelding, omdat transactie B probeert de data te wijzigen die transactie A net gewijzigd heeft. Omdat transactie A nog niet gecommit heeft, heeft A namelijk nog een write-lock op die data.

Opmerking: met data wordt in dit geval de rij bedoeld die door transactie A gewijzigd is. Het gaat dus niet om een lock op een hele tabel. U kunt dit nagaan door straks bij onderdeel b als alternatief voor transactie B de opdracht

```
update Bezoek
set    verblijfsduur = 5
where  reis = 32
       and volgnr = 1
```

te nemen: die update wordt gewoon uitgevoerd.

Opmerking: het veroorzaken van een update-conflict is de enige manier die we hebben om een lost-update-scenario te simuleren. Lost updates mogen immers bij geen enkel isolation level voorkomen, en we kunnen een transactie niet starten met 'no isolation level'.

b Op het moment dat transactie B probeert zijn update uit te voeren in IQU2 wordt de volgende foutmelding gegeven:

```
lock conflict on no wait transaction
deadlock
update-conflicts with concurrent update
concurrent transaction number is 36.
```

De derde regel geeft aan wat er aan de hand is: de update van transactie B conflicteert met de concurrente update van transactie A. Het rdbms weet dat A een lock heeft op de data die B wil updaten en meldt dus een 'lock conflict'.

Opmerking: op de tweede regel van de foutmelding staat de term 'deadlock'. Dit is een beetje verwarrend, omdat het hier niet gaat om een 'totale' deadlock, waarin *alle* transacties op elkaar staan te wachten en er dus helemaal niets meer kan gebeuren. Het gaat hier om twee transacties, waarvan de tweede een halt wordt toegeroepen omdat die probeert gegevens te updaten waarop de eerste al een update heeft uitgevoerd; de tweede wordt dus afgebroken, maar de eerste kan gewoon verdergaan.

Opmerking: als u de code van transactie B verandert in

```
delete from Bezoek
where       reis = 31 and volgnr = 1
```

krijgt u hetzelfde resultaat: een lock conflict.

15.7 Een scenario dat een non-repeatable read voor transactie A veroorzaakt is dat van tabel 15.15.

Tabel 15.16 Transactie A doet een non-repeatable read

transactie A	*transactie* B
`select *` `from Transport;`	

	`update Transport`
	`set omschrijving = 'upbeamen'`
	`where code = 'BU';`
	`commit`

`select *` `from Transport`	

Vergeet de commit van transactie B niet!
We gaan weer uit van een IQU1 met gebruiker Ruimtereisbureau voor transactie A en een IQU2 met gebruiker Sysdba voor transactie B. Transactie A moet natuurlijk isolation level read committed hebben om de verandering van B te kunnen zien:

```
set transaction read only no wait
  isolation level read committed record_version --fn IQU1
```

Transactie B kan gewoon de default transactieopties gebruiken.
De eerste select van transactie A geeft dan als resultaat:

```
CODE   OMSCHRIJVING
====== ============
RV     ruimteveer
BU     beam up
```

De tweede select geeft zoals verwacht:

```
CODE   OMSCHRIJVING
====== ============
RV     ruimteveer
BU     upbeamen
```

15.8 Een scenario dat een phantom row voor transactie A veroorzaakt is:

Tabel 15.17 Transactie A ziet een phantom row verschijnen

transactie A	*transactie* B

`select duur`
`from Reis`
`where duur > 500;`	

	`update Reis`
	`set duur = duur * 10;`
	`commit`

`select duur`
`from Reis`
`where duur > 500`	

Vergeet de `commit` van transactie B niet!
We gaan weer uit van een IQU1 met gebruiker Ruimtereisbureau voor transactie A en een IQU2 met gebruiker Sysdba voor transactie B. Transactie A moet natuurlijk isolation level read committed hebben om de verandering van B te kunnen zien:

```
set transaction read only no wait
  isolation level read committed record_version --fn IQU1
```

De eerste `select` van transactie A geeft dan als resultaat:

```
    DUUR
=========
     1380
     1340
```

De tweede `select` geeft zoals verwacht iets anders:

```
    DUUR
=========
     3900
    13800
     3800
    13400
```

15.9a Transactie B doet bij zijn tweede `select` een dirty read, omdat transactie A zijn `insert` nog niet gecommit heeft.
Het gaat duidelijk niet om een lost update (van transactie A), want er wordt geen `update`-statement uitgevoerd (het is ook geen lost insert, want de `insert` van A is het enige statement in het gehele scenario dat iets verandert in de database).

Het is geen non-repeatable read van transactie B, omdat A nog niet gecommit heeft. (En als A wel gecommit zou hebben voor B's tweede `select` was het ook geen non-repeatable read, omdat alle rijen die B tijdens zijn eerste `select` las tijdens de tweede select nog aanwezig en onveranderd zijn).

Het is geen phantom, omdat A nog niet gecommit heeft.

b We willen dat transactie B niet de kans loopt een dirty read te doen, dus we moeten ervoor zorgen dat B alleen gecommitte gegevens van andere transacties kan lezen:

```
set transaction read only no wait
  isolation level read committed record_version -- voor B
```

Voor transactie A maakt het niet uit; als het erom gaat een zo laag mogelijk isolation level te kiezen kunnen we voor A het volgende nemen:

```
set transaction read write no wait
  isolation level read committed record_version -- voor A
```

15.10 Een scenario waarin transactie A een dirty read doet vanwege een `insert` van transactie B is het volgende:

Tabel 15.18 Transactie A doet een dirty read

transactie A	*transactie* B

	`insert into Klant`
	`values (1460,`
	`'Wilson', 1452)`

`select naam`
`from Klant`
`where aanbrenger =`
`1452`	

Opmerking: natuurlijk kan A die dirty read alleen daadwerkelijk doen bij geen of een laag genoeg isolation level; in Firebird kunnen we dit dus niet simuleren.

15.11 Een scenario waarin transactie A een non-repeatable read doet vanwege een delete van transactie B is het volgende:

Tabel 15.19 Transactie A doet een non-repeatable read

transactie A	*transactie* B
`select *` `from Klacht;`	

	`delete from Klacht`
	`where nr = 2;`
	`commit`

`select *` `from Klacht`	

Opmerking: Natuurlijk kan A die non-repeatable read alleen doen bij geen of een laag genoeg isolation level; in Firebird kunnen we dit scenario simuleren met isolation level `read committed record_version` voor transactie A.

15.12 Een scenario waarin transactie A een inconsistent analysis doet vanwege een `delete` van transactie B is het volgende:

Tabel 15.20 Transactie A doet een inconsistent analysis

transactie A	*transactie* B

`select klant,`
`count(nr)`
`from Order_`
`group by klant;`	

	`delete from Order_`
	`where klant = 1234;`
	`commit`

`select klant,`
`count(nr)`
`from Order_`
`group by klant`	

Opmerking: natuurlijk kan A die inconsistent analysis alleen doen bij geen of een voldoende laag isolation level; in Firebird kunnen we dit scenario simuleren met isolation level `read committed record_version` voor transactie A.

Uitwerkingen hoofdstuk 16

16.1 t/m **16.4** Geen uitwerking.

16.5 Code voor de tweede trigger:

```
set term ^;
create trigger tIngredient_ad for Ingredient after delete
as begin
   execute procedure pGerecht_update_energiePP(old.gerecht);
   end^
set term ;
```

16.6 t/m **16.8** Geen uitwerking.

16.9 De benodigde trigger is een before-delete-trigger:

```
set term ^;
create trigger tGerecht_bd_cascading_delete
for Gerecht before delete
as begin
   delete
   from  Ingredient
   where gerecht = old.naam;
   end^
set term ;
```

De triggernaam is samengesteld volgens de gebruikelijke conventie, met een extra, verduidelijkend achtervoegsel.

16.10 en **16.11** Geen uitwerking.

16.12b De regel kan niet worden overtreden bij invoer van een nieuwe cursus (insert op Cursus), maar wel bij een update, indien de examinator daarbij wordt gewijzigd. Bij een delete op Cursus (gevolgd door een cascading delete van bijbehorende Begeleider-regels) is geen overtreding mogelijk.
Ook invoer van een nieuwe begeleider (insert op Begeleider) kan tot overtreding leiden, maar een delete niet.

c Op basis van het voorgaande zullen we twee triggers schrijven:
- trigger 1: een before-update-trigger op Cursus
- trigger 2: een before-insert-trigger op Begeleider

Beide laten we een exception aanroepen, met de volgende code:

```
create exception eExaminatorIsBegeleider 'Wanneer een docent examinator is van een cursus mag hij geen begeleider
zijn van die cursus.'
```

De triggers ontwikkelen we in stappen.

Trigger 1, eerste opzet:

```
create trigger tCursus_bu
for Cursus before update
as begin
   if (de examinator is veranderd
       and de nieuwe examinator is begeleider van deze cursus)
      then exception eExaminatorIsBegeleider;
   end
```

Dit geeft:

```
set term ^;
create trigger tCursus_bu
for Cursus before update
as begin
   if (new.examinator <> old.examinator
       and new.examinator in (select docent
                              from   Begeleider
                              where  cursus = new.code))
      then exception eExaminatorIsBegeleider;
   end^
set term ;
```

De tweede trigger krijgt de volgende opzet:

```
create trigger tBegeleider_bi
for Begeleider before insert
as begin
   if (de nieuwe begeleider is examinator van de betreffende cursus)
      then exception eExaminatorIsBegeleider;
   end^
```

Dit leidt tot:

```
set term ^;
create trigger tBegeleider_bi
for Begeleider before insert
as begin
   if (new.docent = (select examinator
                     from   Cursus
                     where  code = new.cursus))
      then exception eExaminatorIsBegeleider;
   end^
set term ;
```

16.13a SQL-statement:

```
create exception eOrderwijziging
'Datum verstreken; niet mogelijk nog wijzigingen aan te brengen'
```

b We geven eerst een oplossing met drie single-event-triggers:
– een trigger die voorkomt dat aan een oude order nog een orderregel wordt toegevoegd
– een trigger die voorkomt dat uit een oude order een orderregel wordt verwijderd
– een trigger die voorkomt dat van een oude order een orderregel een update ondergaat.

De drie triggers in één script:

```
set term ^;
create trigger tOrderregel_bi
for Orderregel before insert
as begin
   if (current_date <> (-- datum van order bij in te voegen orderregel
                        select datum
                        from   Order_
                        where  nr = new.order_))
      then exception eOrderwijziging;
   end^

create trigger tOrderregel_bd
for Orderregel before delete
as begin
   if (current_date <> (-- datum van order bij te verwijderen orderregel
                        select datum
                        from   Order_
                        where  nr = old.order_))
      then exception eOrderwijziging;
   end^
```

```
create trigger tOrderregel_bu
for Orderregel before update
as begin
   if (current_date <> (-- datum van order bij aan te passen orderregel
                        select datum
                        from   Order_
                        where  nr = old.order_))
      then exception eOrderwijziging;
   end^
set term ;^
```

Merk op dat de code van de triggers tOrderregel_bd en tOrderregel_bu identiek is. We kunnen deze samenvoegen tot één multi-event-trigger van het type 'delete or update':

```
set term ^;
create trigger tOrderregel_bdu
for Orderregel before delete or update
as begin
   if (current_date <> (select datum
                        from   Order_
                        where  nr = old.order_))
      then exception eOrderwijziging;
   end^
set term ;^
```

Er zijn nog tal van varianten. We geven er één, waarbij we ons beperken tot trigger tOrderregel_bi, waarin de datum in een variabele wordt opgeslagen:

```
set term ^;
create trigger tOrderregel_bi
for Orderregel before insert
as declare variable v_orderdatum date;
   begin
   select datum
   from   Order_
   where  nr = new.order_
   into   v_orderdatum;

   if (v_orderdatum <> current_date)
      then exception eOrderwijziging;
   end^
set term ;^
```

Door gebruik te maken van boolean systeemvariabelen `inserting`, `deleting` en `updating` kunnen we zelfs alle (before-) triggers op Orderregel onderbrengen in één trigger tOrderregel_b**idu**, van het type 'insert or delete or update', als volgt (in een versie met variabele).

```
set term ^;
create trigger tOrderregel_bidu
for Orderregel before insert or delete or update
as declare variable v_orderdatum date;
   begin
   if (inserting)
   then select datum
        from   Order_
        where  nr = new.order_
        into   v_orderdatum;

   if (deleting or updating)
   then select datum
        from   Order_
        where  nr = old.order_
        into   v_orderdatum;

   if (v_orderdatum <> current_date)
      then exception eOrderwijziging;
   end^
set term ;^
```

c Het moet ook onmogelijk zijn om de orderdatum te veranderen. Want anders is er de volgende sluiproute: orderdatum veranderen in de huidige datum, orderregelverzameling aanpassen, orderdatum weer terugzetten.

16.14a Code voor procedure pUpdateTotaalbedragOrder:

```
set term ^;
create procedure pUpdateTotaalbedragOrder(p_ordernr integer)
as begin
   update Order_
   set    totaalbedrag = (select sum(bedrag)
                          from   Orderregel
                          where  order_ = :p_ordernr)
   where  nr = :p_ordernr;
end^
set term ;
```

Testen:

```
execute procedure pUpdateTotaalbedragOrder(…)
```

Vul op de plaats van … een ordernummer in.

b Het ordertotaalbedrag moet worden herberekend bij de volgende events:
- after insert op Orderregel
- after delete op Orderregel
- after update op Orderregel

Opmerkingen
- Daarnaast is het denkbaar dat een ordertotaalbedrag 'zomaar' wordt gewijzigd. In plaats van dit toe te staan en 'after update' te corrigeren, is het beter dit te voorkomen, met een before-update-trigger en een exception.
- We nemen aan dat het wijzigingen van Orderregel.order niet kan voorkomen.

16.15a Er wordt hier een after-insert-trigger voor Orderregel gevraagd. Om naamsconflicten met eventuele eerder gemaakte after-update-triggers voor Orderregel te vermijden, zullen we de triggernaam van een extra achtervoegsel _voorraad voorzien:

```
set term ^;
create trigger tOrderregel_ai_Voorraad
for Orderregel after insert
as begin
   update Artikel
   set    voorraad = voorraad - new.aantal
   where  nr = new.artikel;
end^
set term ;
```

b Definitie van exception:

```
create exception eVoorraad_negatief
   'Een voorraad mag niet negatief zijn'
```

De definitie van de nieuwe trigger bouwen we op in stappen.

Stap 1

```
create or alter trigger tOrderregel_bi_voorraad
for Orderregel before insert
as begin
   bepaal de voorraad van het artikel van de nieuwe orderregel;
   if (voorraad niet-toereikend)
      then exception eVoorraad_negatief;
   werk voorraad bij;
end^
```

Stap 2
De voorraad moet kennelijk na het bepalen ervan worden onthouden. Daar is een variabele voor nodig. Verder is het mooier de stap 'werk voorraad bij' onder te brengen in een `else`-clausule van het `if`-statement:

```
create or alter trigger tOrderregel_bi_voorraad
for Orderregel before insert as
declare variable v_voorraad integer;
begin
   bepaal de voorraad van het artikel van de nieuwe orderregel
   en ken deze toe aan v_voorraad;
   if (v_voorraad < bestelde aantal)
      then exception eVoorraad_negatief;
   else
      werk voorraad bij;
end^
```

Stap 3 (eindoplossing)

```
set term ^;
create or alter trigger tOrderregel_bi_voorraad
for Orderregel before insert as
declare variable v_voorraad integer;
begin
   -- bepaal voorraad en ken deze toe aan v_voorraad
   select voorraad
   from   Artikel
   where  nr = new.artikel
   into   v_voorraad;

   if (v_voorraad < new.aantal)
      then exception eVoorraad_negatief;
   else
      -- werk voorraad bij
      update Artikel
      set    voorraad = voorraad - new.aantal
      where  nr = new.artikel;
end^
set term ;
```

16.16 Wanneer we alleen daadwerkelijk aangebrachte veranderingen op tabel Artikel loggen, moeten we drie after-triggers schrijven op Artikel: een after insert, een after delete en een after update. Hier volgen de drie triggerdefinities in één script:

```
set term ^;

create trigger tArtikel_ai
for Artikel after insert
as begin
   insert into LogOrderdatabase
      values (user, current_date, 'Artikel', 'I');
   end^

create trigger tArtikel_ad
for Artikel after delete
as begin
   insert into LogOrderdatabase
      values (user, current_date, 'Artikel', 'D');
   end^

create trigger tArtikel_au
for Artikel after update
as begin
   insert into LogOrderdatabase
      values (user, current_date, 'Artikel', 'U');
   end^

set term ;
```

Desgewenst kunnen we deze tot één multi-event-trigger combineren, als volgt.

```
set term ^;
create trigger tArtikel_aidu
for Artikel after insert or delete or update
as begin
   if (inserting)
      then insert into LogOrderdatabase
           values (user, current_date, 'Artikel', 'I');
      else if (deleting)
              then insert into LogOrderdatabase
                   values (user, current_date, 'Artikel', 'D');
              else -- updating
                   insert into LogOrderdatabase
                   values (user, current_date, 'Artikel', 'U');
   end^
set term ;
```

In de laatste `else`-clausule hadden we in plaats van het commentaar '`-- updating`' ook '`if (updating) then`' kunnen schrijven. Maar zo'n extra `if` is niet nodig. Immers, de laatste `else`-clausule wordt alleen maar uitgevoerd wanneer `inserting` en `deleting` beide *false* zijn, en dan is `updating` vanzelf *true*.

Uitwerkingen hoofdstuk 17

17.1 Geen uitwerking.

17.2 SQL-statement:

```
insert into Rdb$relations (rdb$relation_name)
values ('Test')
```

NB We willen straks de veranderingen ongedaan maken door middel van een rollback, maar mocht u nu toch een commit willen proberen, dan is dit het resultaat:

```
unsuccessful metadata update
TABLE Test
Can't have relation with only computed fields or constraints.
```

We kunnen dus niet écht een tabel zonder kolommen toevoegen.

17.3 Geen uitwerking.

17.4 Inspectie van Rdb$database:

```
select *
from   Rdb$database
```

geeft als resultaat:

```
RDB$DESCRIPTION RDB$RELATION_ID RDB$SECURITY_CLASS RDB$CHARACTER_SET_NAME
=============== ============== ================== ======================
<NULL>                     131 <NULL>             NONE
```

a In de kolom rdb$relation_id wordt het eerstvolgende vrije 'id' voor gebruikerstabellen (en -views) bewaard. Dit blijkt uit inspectie van de tabel Rdb$relations die ook zo'n id-kolom blijkt te hebben.

b De kolom rdb$character_set_name bevat de naam van een character set (zie paragraaf 3.2 van leereenheid 7), indien Firebird een andere character set gebruikt dan die van het besturingssysteem.

c Er is geen kolom voor de databasenaam omdat het 'single point of definition' hiervan de fdb-bestandsnaam is. Deze is vastgelegd op het niveau van het besturingssysteem.

17.5 Waar nu de kunstmatige domeinnaam Rdb$1 staat (in elke tabel één keer), zou Datum staan. Verder zou het volgnummer in eventuele andere kunstmatige domeinnamen anders luiden.

17.6 SQL-statement:

```
select rdb$relation_name, rdb$field_name, rdb$field_position
from   Rdb$relation_fields
where  rdb$system_flag = 0
```

17.7 In één script:

```
insert into Rdb$relations
(rdb$relation_name, rdb$owner_name, rdb$system_flag)
values ('TEST', 'RUIMTEREISSIMPEL', 0);

insert into Rdb$fields
(rdb$field_name, rdb$field_type, rdb$field_length,
 rdb$system_flag)
values ('RDB$101', 35, 8, 0);

insert into Rdb$fields
(rdb$field_name, rdb$field_type, rdb$field_length,
 rdb$system_flag)
values ('RDB$102', 14, 1, 0);
```

```
insert into Rdb$relation_fields
(rdb$relation_name, rdb$field_name, rdb$field_source,
 rdb$field_position, rdb$system_flag)
values ('TEST', 'KOLOM1', 'RDB$101', 0, 0);

insert into Rdb$relation_fields
(rdb$relation_name, rdb$field_name, rdb$field_source,
 rdb$field_position, rdb$system_flag)
values ('TEST', 'KOLOM2', 'RDB$102', 1, 0);

commit
```

Vervolgens:

```
insert into Test values (current_date, 'a')
```

Met een

```
select * from Test
```

controleren we vervolgens of het gelukt is, met als resultaat:

```
KOLOM1              KOLOM2
=================== ======
08-feb-2012 00:00:00 a
```

17.8 SQL-statement voor alle recursieve foreign key-constraints:

```
select Refc.rdb$constraint_name
from   Rdb$Ref_constraints Refc
       join Rdb$relation_constraints Relc1
         on Refc.rdb$constraint_name = Relc1.rdb$constraint_name
       join Rdb$relation_constraints Relc2
         on Refc.rdb$constraint_name = Relc2.rdb$constraint_name
where  Relc1.rdb$relation_name = Relc2.rdb$relation_name
```

17.9 Eerst raadplegen we de tabel Rdb$generators om de naam van de sequence te achterhalen die de indexvolgnummers bijhoudt. Dit blijkt Rdb$index_name te zijn.
Vervolgens vragen we aan de sequence Rdb$index_name wat zijn laatst uitgedeelde nummer is:

```
select next value for rdb$index_name
from   Rdb$database
```

Toelichting: als brontabel kozen we Rdb$database, omdat deze maar één rij heeft en in elke data dictionary zit. Een nadeel van deze oplossing is dat hij een neveneffect heeft: het gezochte nummer wordt opgehoogd.

17.10 SQL-statement:

```
create view vRelations (relation_name, owner_name, system_flag)
as select rdb$relation_name, rdb$owner_name, rdb$system_flag
   from   Rdb$relations
```

17.11a Equivalent `select`-statement:

```
select rdb$relation_name
from   Rdb$relations
where  rdb$system_flag = 0
```

b Uitgaande van RuimtereisSimpel toont het volgende select-statement de kolomnamen van tabel Reis:

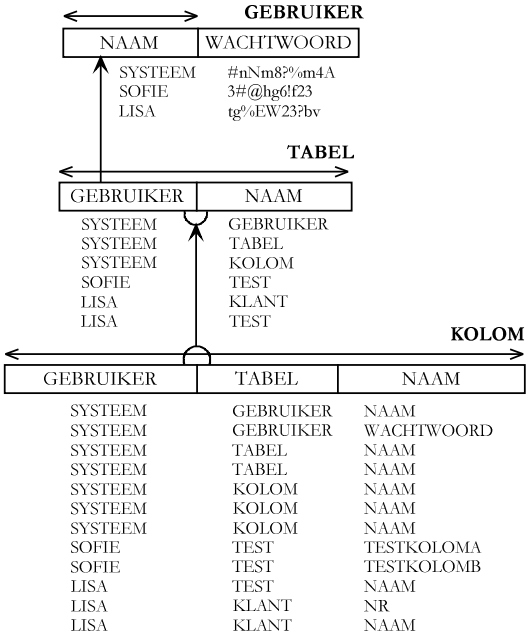
```
select rdb$field_name
from   Rdb$relation_fields
where  rdb$relation_name = 'REIS'
       and rdb$system_flag = 0
```

17.12 `drop table` wordt `drop view`. Verder kan de eerste selectieconditie worden weggelaten, omdat de data dictionary zelf geen gebruikersviews bevat. Omkeren van de tweede conditie geeft het verlangde resultaat:

```
select 'drop view ' || trim(rdb$relation_name) || ';'
from   Rdb$relations
where  rdb$view_blr is not null
```

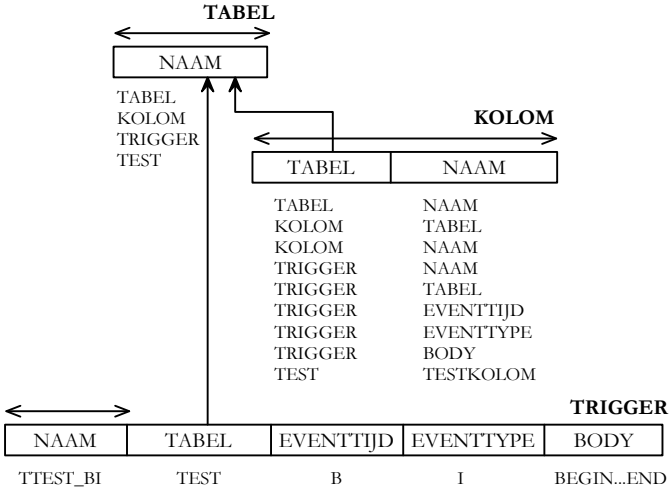
17.13 Een oplossing geeft het strokendiagram van figuur 17.11, met de drie gevraagde 'kern-metatabellen'.

Toelichting
- De tabel Gebruiker bevat naast de 'supergebruiker' Systeem twee gewone gebruikers: Sofie en Lisa. Sofie is eigenaar van één tabel, genaamd Test (met twee kolommen). Lisa is eigenaar van één tabel, die ook Test heet (met één kolom), en van een tweede tabel, die Klant heet (met twee kolommen).
- De tabel Tabel bevat de drie metatabellen en de drie gebruikerstabellen.
- De tabel Kolom bevat de zeven kolommen van de data dictionary en de vijf kolommen van de gebruikerstabellen.
- We zijn uitgegaan van de aanname dat namen van databaseobjecten in de data dictionary in hoofdletters worden opgeslagen.



Figuur 17.11 Strokendiagram metatabellen

17.14 Zie figuur 17.12.



Figuur 17.12 Populatiediagram