I N L E I D I N G D A T A B A S E S

1986/1987

a235

Dit dictaat is gebaseerd op het dictaat Databases (a134), dat werd samengesteld door Ir. A. van der Ende en Ir. J.W.M. Langendoen.
Bij de totstandkoming van het dictaat voor a235 werd ook medewerking verleend door Ir. J.H. ter Bekke.
Tekstverwerking: Mirjam Koetsier.

oktober 1986.

TECHNISCHE UNIVERSITEIT DELFT
Faculteit der Technische
Wiskunde en Informatica
Vakgroep Informatiesystemen
Julianalaan 132
2628 BL DELFT

INHOUD blz.

1. OBJECTIEVEN VAN EEN DATABASE

1.1. Omgeving

Binnen organisaties en bedrijven vinden allerlei activiteiten plaats. In de informatiebehoeften van deze activiteiten kan voorzien worden door een informatiesysteem. Een informatiesysteem is te beschouwen als het geheel van gegevensverwerkende activiteiten en omvat o.a.:

- een database (of één of meer bestanden);
- programma's voor het verwerken van invoergegevens;
- procedures uitgevoerd door mensen met als resultaat invoergegevens;
- programma's voor het verkrijgen van uitvoergegevens;
- procedures uitgevoerd door mensen op basis van de uitvoergegevens;
- organisatie van hard- en software;
- organisatie van mensen.

Het opzetten van een database is dus niet een op zichzelf staande activiteit maar maakt deel uit van het ontwikkelen van een informatiesysteem en is bedoeld om te voorzien in informatiebehoeften.

1.2. Wat is een database?

Een database kan worden gezien als een (grote) verzameling records opgeslagen op achtergrondgeheugen. Binnen deze verzameling zijn weliswaar aparte deelverzamelingen - de records van één recordtype - te onderscheiden maar deze staan echter niet volstrekt los van elkaar, omdat ook de samenhang tussen records is vastgelegd. Hierdoor kan meervoudige opslag (data redundancy) van dezelfde samenhangende gegevens worden vermeden. We zeggen dan dat de database "integrated" is.

Tot de records hebben diverse applicaties op verschillende manier toegang. Onderscheiden worden "on-line" en "batch" applicaties. Een "on-line" applicatie staat in rechtstreekse verbinding met de database, en krijgt per ommegaande antwoord op de gestelde vragen. Een "batch" applicatie heeft geen rechtstreekse verbinding met de database maar verwerkt met behulp van programma's in één keer een groot aantal gegevens op een tijdstip dat binnen

zekere grenzen door het systeem wordt bepaald.

Iedere applicatie heeft toegang tot slechts een deel van de opgeslagen records. Geen enkele applicatie heeft het alleenrecht op zijn records. Alle records zijn in principe toegankelijk voor alle applicaties. We zeggen dat de database "shared" is.

Een voorbeeld van een database met bijbehorende applicaties. Een bedrijf dat grondstoffen verwerkt tot produkten legt in een database de voorraad van elke grondstof vast. Om de voorraad aan te vullen worden orders geplaatst waarvoor ook gegevens zoals hoeveelheid van de betreffende grondstof en vermoedelijke afleverdatum worden opgeslagen. Als de gegevens van een grondstof ook weer bij een order worden opgenomen ontstaat data overtolligheid (data redundancy). Om dit te vermijden zullen alleen records van het type grondstof en van het type order in de database worden vastgelegd. Natuurlijk moet dan ook de samenhang tussen de records - bij één order hoort één bepaalde grondstof - worden opgenomen.

Een typische "batch" applicatie is het maken van een produktieplanning, waarbij wordt nagegaan of er voldoende voorraden van de gewenste grondstoffen zijn. Een typische "on-line" applicatie is het opvragen van de afleverdatum van een order.

1.3. Bestanden en hun problemen

Databases zijn er niet plotseling gekomen. Zij zijn een logische voortzetting van bestanden en geven een oplossing voor de problemen die ontstaan bij het gebruik van bestanden. Een bestand wordt gekenmerkt door de volgende eigenschappen:

1. het is een verzameling records van één recordtype;
2. er is maar één applicatie, zodat alle programma's op dezelfde manier toegang hebben tot precies dezelfde gegevens;
3. de opslagstructuur van en de toegangsmogelijkheden tot het bestand zijn afgestemd op de applicatie zodanig dat de verwerkingstijd zo klein mogelijk is;
4. de logica van de programma's is afhankelijk van de opslagstructuur van en/of de toegangsmogelijkheden tot het bestand.

Problemen met bestanden gaan zich voordoen als gevolg van de groei in omvang of de groei in verscheidenheid.

De groei in omvang - toename van het aantal records in het bestand - kan leiden tot een verslechtering van de prestatie hetgeen zich uit in een verlaging van de verwerkingssnelheid. Daalt de snelheid onder een zeker minimum dan zal gezocht worden naar een andere opslagstructuur of toegangsmogelijkheid waardoor de verwerkingssnelheid weer toeneemt. De ingrijpende gevolgen van zo'n verandering zijn:

1. laden van het bestand volgens de nieuwe structuur;
2. herschrijven van alle programmatuur daar de logica hiervan was afgestemd op de structuur van het bestand.

Een voorbeeld van zo'n verandering is de overgang van een sequentieel naar een index-sequentieel georganiseerd bestand.

De groei in verscheidenheid - toename van het aantal applicaties - leidt tot een uitbreiding van het bestand als nieuwe gegevenstypen nodig zijn. Deze uitbreiding kan op twee manieren worden gerealiseerd, n.l.:

1. het bestaande bestand wordt uitgebreid met de noodzakelijke nieuwe gegevens. Zowel de bestaande als de nieuwe applicatie hebben toegang tot het nieuwe bestand
2. het bestaande bestand blijft ongewijzigd. De noodzakelijke nieuwe gegevens en de relevante bestaande gegevens worden samengevoegd tot een nieuw bestand. De bestaande applicatie heeft alleen toegang tot het bestaande bestand. De nieuwe applicatie heeft alleen toegang tot het nieuwe bestand.

Aan beide realisaties kleven bezwaren die aan de hand van het volgende voorbeeld zullen worden beschouwd.

Een produktieonderneming heeft een geautomatiseerde salarisadministratie ten behoeve waarvan een bestand is ontworpen dat van ieder personeelslid de volgende gegevens bevat:

personeelsnummer
naam
adres
woonplaats
functie
geboortedatum
salaris
burgelijke staat
aantal kinderen

De onderneming beschikt over een eigen medische dienst die op zekere dag constateert dat de personeelsleden medisch gezien een verschillend risico hebben, afhankelijk van leeftijd, funktie, de afdeling waar men werkt en de huidige lichamelijke en geestelijke gezondheid. Aangezien voorkomen beter is dan genezen, wil men de personeelsleden periodiek oproepen voor een medisch onderzoek. Men besluit hiervoor een geautomatiseerd systeem te ontwerpen waarvoor de volgende gegevens nodig zijn:

personeelsnummer
naam
adres
woonplaats
functie
afdeling
geboortedatum
lichamelijke gezondheidstoestand
geestelijke gezondheidstoestand

Er zijn twee mogelijkheden.

1. Het oorspronkelijke bestand uitbreiden met de nieuwe gegevens, zodat het bestand dan de volgende gegevens gaat bevatten:

personeelsnummer
naam
adres
woonplaats
geboortedatum
functie
afdeling
salaris
burgelijke staat
aantal kinderen
lichamelijke gezondheidstoestand
geestelijke gezondheidstoestand

De nadelen van deze realisatie zijn:
- de programmatuur t.b.v. de salarisadministratie moet worden gewijzigd en getest;
- de gegevens zijn niet beschermd daar beide applicaties toegang hebben tot alle gegevens. Zo kan de salarisadministratie beschikken over gegevens die betrekking hebben op de lichamelijke en geestelijke gezondheidstoestand van een personeelslid;
- beide applicaties hebben op dezelfde manier toegang tot het bestand. De salarisadministratie zal echter in het algemeen toegang willen verkrijgen d.m.v. het personeelsnummer terwijl de medische dienst d.m.v. de combinatie geboortedatum, functie en afdeling zal willen selecteren;
- uitbreiding met nog een applicatie zal over het algemeen niet mogelijk zijn omdat nu twee afdelingen ieder met eigen programmatuur zich zullen moeten aanpassen;
- de continuïteit van de werkzaamheden is niet gewaarborgd. Tot een zekere datum wordt met het oude bestand gewerkt, daarna met het nieuwe. De invoering van het nieuwe bestand vergt dan ook veel organisatorisch talent, maar kan niettemin een (korte) periode van chaos met zich meebrengen.

2. De tweede mogelijkheid is de twee bestanden naast elkaar te gebruiken. Eén bestand ten behoeve van de salarisadministratie (best 1) en één ten behoeve van de medische dienst (best 2), met resp. de volgende gegevens:

best 1	best 2
personeelsnummer	personeelsnummer
naam	naam
adres	adres
woonplaats	woonplaats
geboortedatum	geboortedatum
functie	functie
salaris	afdeling
burgelijke staat	lichamelijke gezondheidstoestand
aantal kinderen	geestelijke gezondheidstoestand

De nadelen zijn:

- de waarde van de gegevens die in beide bestanden voorkomen moet voortdurend gelijk worden gehouden. In het algemeen zijn hiervoor geen programmatische hulpmiddelen beschikbaar. Alleen het door mensen juist uitvoeren van procedurele regels is een waarborg voor het gelijk zijn van de waarden van meer dan éénmaal opgeslagen gegevens. Het is dan ook niet verwonderlijk dat het kan voorkomen dat volgens het ene bestand een personeelslid op een ander adres woont dan volgens het andere bestand. In sommige situaties kan het zeer moeilijk zijn bij een verschil de juiste waarde van een gegeven vast te stellen.
  Het probleem van gegevens met tegenstrijdige inhoud staat bekend onder de naam data inconsistency;
- het nieuwe bestand moet worden opgebouwd uit het bestaande bestand. De programmatuur hiervoor moet apart worden geschreven en getest;
- het meer dan één keer opslaan van dezelfde gegevens kost extra achtergrondgeheugen. Gezien de huidige capaciteit van achtergrondgeheugens heeft dit bezwaar veel minder gewicht gekregen.

1.4. Belangrijkste doelstellingen

Het database concept is er gekomen als antwoord op de in de vorige paragraaf genoemde problemen. De belangrijkste doelstellingen zijn:

1. het kunnen vermijden van data redundancy en data inconsistency;
2. het nastreven van data onafhankelijkheid te weten:
   a. logische data onafhankelijkheid (logical data independence). Hieronder wordt verstaan de onafhankelijkheid van de programmatuur voor wijzi-

gingen in de logische struktuur van de database. Voorbeelden van dit soort wijzigingen zijn: het verwijderen of toevoegen van een record-type, het uitbreiden van één of meer velden, het aanbrengen van nieuwe relaties, etc.

Een gevolg van de logische data onafhankelijkheid is dat een applicatie slechts tot een deel van de gegevens in de database toegang heeft en van de rest is afgeschermd.

b. fysieke data onafhankelijkheid (physical data independence). Hieronder wordt verstaan de onafhankelijkheid van de programmatuur voor wijziging in de opslagstructuur van of de toegangsmogelijkheden tot de gegevens. De applicatie is dan afgeschermd van allerlei fysieke zaken, zodat hierin wijzigingen kunnen worden aangebracht zonder dat aanpassing van de programmatuur noodzakelijk is.

2. ARCHITECTUUR VAN EEN DATABASE SYSTEEM

2.1. Beschrijven van een database

Het werken met een bestand vereist een beschrijving van het bestand (denk hierbij aan de Data Division in een Cobolprogramma), die deel uitmaakt van ieder programma en o.a. bevat:

- een recordbeschrijving: de velden waaruit het record bestaat en van ieder veld het formaat;
- de blokgrootte: het aantal records in één fysieke eenheid op achtergrondgeheugen;
- de organisatie: sequentieel, index-sequentieel, direct en zonodig de keyvelden.

Het werken met een database verschilt in wezen niet van het werken met een bestand. Het vereist ook een (database)beschrijving die echter niet in het programma wordt opgenomen, daar dan zeker niet aan de doelstelling van logische data onafhankelijkheid zou worden voldaan. Iedere applicatie zou dan toegang hebben tot de volledige database. Omdat een applicatie slechts toegang heeft tot een deel van de database, moet het mogelijk zijn een deelbeschrijving te kunnen geven. Uiteraard is een volledige beschrijving ook nodig. Daarom zijn de volgende begrippen gedefinieerd:

Conceptual Schema: beschrijving van de gehele database; het bevat beschrijvingen van alle gegevenstypen en van de relaties daartussen.

External Schema: beschrijving van dat gedeelte van de database waartoe een applicatie toegang heeft; het bevat beschrijvingen van gegevenstypen en van de relaties daartussen.

Per database is er één conceptual schema en voor iedere applicatie één external schema. Het conceptual schema is een vereniging van alle external schema's, in die zin dat ieder veld dat in één of ander external schema is beschreven ook weer terug te vinden is in het conceptual schema. Gegevenstypen in een external schema behoeven daarentegen niet gelijk te zijn aan gegevenstypen in het conceptual schema, d.w.z. zij behoeven niet uit dezelfde velden te bestaan.

Door deze scheiding in conceptual- en external schema kan een uitbreiding van de database tot stand worden gebracht zonder wijziging van de programmatuur van de bestaande applicaties.

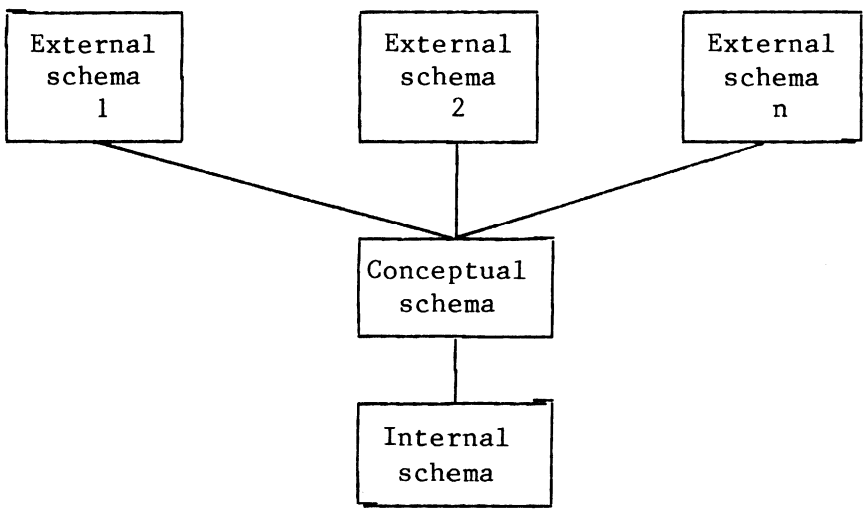
Zoals uit het voorafgaande blijkt is de beschrijving van de fysieke aspekten van de database noch in het external noch in het conceptual schema opgenomen, daar anders zeker niet aan de doelstelling van fysieke data onafhankelijkheid kan worden voldaan. Uiteraard moeten deze aspekten wel worden beschreven en daarom is ook nog een internal schema gedefinieerd:

Internal Schema: de fysieke beschrijving van de gehele database, o.a. opslagstruktuur en toegangsmogelijkheden (index, direct) van ieder internal record, blokgrootte.

Het essentiele verschil tussen het internal en het conceptual schema is dus dat in het conceptual schema wordt beschreven welke gegevenstypen in de database zijn opgenomen (logische structuur) en in het internal schema hoe de records van deze gegevenstypen worden opgeslagen en op welke wijze deze kunnen worden benaderd.

Door deze scheiding in internal en conceptual schema kan een overgang van b.v. sequentiele naar index-sequentiele toegang tot stand worden gebracht zonder wijziging van het conceptual schema en dus ook zonder wijziging van de bestaande programmatuur.

Schematisch



Een external schema wordt opgesteld voor een applicatie in overleg met de database administrator (zie 2.3.). Het bevat DDL (Data Definition Language) statements, o.a. voor de specificaties van gegevenstypen.

Het conceptual schema wordt opgesteld door de database administrator. Ook dit schema bevat DDL-statements o.a. voor de specificaties van gegevenstypen. Het internal schema wordt opgesteld door de database administrator. Dit schema bevat SDL (Storage Definition Language)-statements om de fysieke eigenschappen van de database te beschrijven.

2.2. DBMS, DDL en DML

Voor het gebruik van databases zijn talloze geautomatiseerde systemen ontwikkeld, die tevens voor het beheer van de databases zorgen. Deze systemen staan bekend onder de naam database management systemen (DBMS). Het zal duidelijk zijn dat om een database management systeem te kunnen gebruiken eerst moet worden bepaald welke gegevens in de database moeten worden vastgelegd en in welke vorm en samenhang. De structuur van de gegevens moet dan zodanig zijn dat ze aan het systeem kan worden aangeboden. Hierbij wordt de bij het database management systeem behorende datataal gebruikt, nl. het deel dat voor de definitie van de gegevens is, de eerder genoemde Data Definition Language (DDL).

Een ander deel van de datataal is bestemd voor het formuleren van de bewerkingen op de gegevens. Net zoals bij het werken met bestanden, zijn er voor het manipuleren met gegevens uit de database opdrachten nodig, b.v. opdrachten voor het lezen en schrijven van gegevens. Omdat in de database ook de samenhang tussen de gegevens is vastgelegd, is het aantal manipulatieopdrachten wel groter dan bij de "klassieke" bestandsorganisatie. Deze opdrachten vormen samen de zgn. Data Manipulation Language (DML).

Globaal kunnen we twee soorten DML's onderscheiden:

- de DML is een zelfstandige taal. Deze taal kan, afhankelijk van de geboden faciliteiten, complex of zeer eenvoudig zijn. In het laatste geval kan hij ook door personen zonder programmeerervaring worden gebruikt om gegevens uit de database op te vragen. Zo'n taal wordt wel een vraagtaal (query language) genoemd.

- de DML-opdrachten zijn ingebed in een bestaande programmeertaal, de zgn. gasttaal. De opdrachten zijn dan uitsluitend voor het verkeer tussen de database en het programma, terwijl bewerkingen op de gegevens in opdrachten van de gasttaal moeten worden beschreven.

Het database management systeem kan, hierbij gebruik makend van de beschrijving van de gegevens - de schema's - de DML-opdrachten afhandelen en maakt daarbij zelf weer gebruik van faciliteiten van het operating systeem.

2.3. De Database Administrator

De doelstellingen die aan het database concept ten grondslag liggen vereisen o.a. dat de gegevens niet meer in de programma's worden gedefinieerd, terwijl de verschillende applicaties toch op een efficiënte wijze met de gegevens moeten kunnen manipuleren. Dit en het feit dat de gegevens gemeenschappelijk worden gebruikt hebben geleid tot een nieuwe functie, die van Database Administrator (DBA). Hoewel deze functie door één persoon of door een groep van personen kan worden vervuld, spreken we verder over de DBA.

De DBA is verantwoordelijk voor het totale beheer van de database. Om een indruk te krijgen van deze functie volgen hier enige taken.

- Het ontwerpen van het conceptuele model. Dit model ontstaat door integratie van de individuele gebruikersbehoeften.

- Het definiëren van de verschillende externe modellen. Uiteraard geschiedt dit in overleg met de projectgroepen van de diverse applicaties.

- Het definiëren van het interne model. Allerlei zaken zoals b.v. opslagstructuur, toegangsmogelijkheden (denk aan indexen), keuze van media, enz. komen hierbij aan de orde.

- Het bewaken van de performance van het systeem. Indien een verandering in het gebruik van de database dit noodzakelijk maakt, moeten wijzigingen worden aangebracht in opslagstructuur, geheugenmedia, etc. (reorganisatie).

- Het voorschrijven van standaards voor documentatie. Denk hierbij aan namen en definities van gegevens.

- Het adviseren van gebruikers bij het ontwerpen van nieuwe toepassingen.

- Het definiëren van een strategie voor back-up en recovery.

- Het definiëren van autorisatie controles. Er moet worden vastgelegd wie met welke gegevens wat mag doen.

- Het treffen van maatregelen om de integriteit van de gegevens in de database zo goed mogelijk te waarborgen.

- Het indien nodig aanpassen van het conceptuele model (herstructurering).

Uit bovenstaande opsomming blijkt dat bij databasetoepassingen in grote organisaties het takenpakket van de DBA zeer omvangrijk is. In de praktijk wordt deze functie dan ook wel gesplitst. Men krijgt dan de functies gegevensbeheer en systeembeheer. De gegevensbeheerder houdt zich vooral bezig met de niet-fysieke zaken (conceptueel en externe modellen, documentatie), terwijl de systeembeheerder voor de fysieke aspecten (DBMS, opslagstructuur, performance) verantwoordelijk is.

2.4. De Data Dictionary

Een belangrijk hulpmiddel voor de database administrator is de Data Dictionary (DD). De DD is in feite niets anders dan een database met gegevens over de gegevens die in de "echte" database zijn opgeslagen of nog opgeslagen moeten worden. Men spreekt dan ook wel over meta-gegevens. Een DD kan een belangrijke rol spelen in de documentatie. Zo kan b.v. van elk gegeven uit de database in de DD worden opgenomen:

- de betekenis en het formaat;
- de herkomst;
- de relatie met andere gegevens;
- in welke programma's (applicaties) het wordt gebruikt;
- wie het mag wijzigen.

Ook de verschillende schema's (conceptueel, extern, intern) kunnen, zowel in de oorspronkelijke als in de vertaalde vorm, in een DD worden opgenomen. Niet altijd zullen alle genoemde mogelijkheden worden geboden.
Het is duidelijk dat een DD zowel bij het ontwerp van een database als bij het gebruik van een database kan worden ingeschakeld.

Afgezien van de schema's waren de bovengenoemde metagegevens ook al van betekenis bij de conventionele (niet database) systemen. Hierbij is vooral te denken aan het belang van het vastleggen van de namen van de gegevens zoals die in de verschillende bestanden voorkomen, met daarbij de vermelding van de juiste betekenis. Dat is ook de reden waarom reeds voor de introductie van DBMS'en bepaalde DD systemen zijn ontwikkeld. Tegenwoordig is de DD meestal geïntegreerd in het DBMS.

Om een goed gebruik van een DD te kunnen maken is de aanwezigheid van een taal vereist, waarmee op eenvoudige wijze de DD kan worden geraadpleegd. Afhankelijk van de aanwezige faciliteiten, zullen dan naast de database administrator ook andere gebruikers van de database de DD als hulpmiddel kunnen inschakelen.

3. DE WERKELIJKHEID IN EEN DATABASE

3.1. Entiteit en object

In een database worden gegevens over objecten vastgelegd die van belang zijn voor het informatiesysteem waarvan de database deel uitmaakt. Afhankelijk van de omgeving kunnen deze objecten zijn: patiënten, studenten, leveranciers, hotelreserveringen, bestellingen, enz.

Van de dingen zoals die in de werkelijkheid bestaan, wordt in het algemeen slechts een beperkt aantal kenmerken beschouwd, n.l. alleen die kenmerken waarin we zijn geïnteresseerd. Zo zullen in een database t.b.v. een studentenadministratie geen gegevens over de haarkleur, de lengte of het gewicht van studenten voorkomen. Het is echter niet denkbeeldig dat in een database t.b.v. de studenten gezondheidszorg wel gegevens als lengte en gewicht zijn opgenomen. Met andere woorden de eigenschappen of kenmerken van de objecten die we willen beschouwen, worden bepaald door de toepassingen.

Het "ding" in de werkelijkheid is veel gecompliceerder dan het "ding" waarvan de gegevens in de database worden vastgelegd. Het ding in de werkelijkheid, vaak entiteit genoemd, is gemodelleerd tot het object dat we voor de betreffende toepassing willen beschouwen.

In de literatuur worden de begrippen entiteit en object vaak door elkaar gebruikt. Wat de één een entiteit noemt, noemt de ander een object en omgekeerd. In dit dictaat zullen we verder alleen spreken over objecten.

3.2. Objecttype en attribuut

De beschrijving van de database zoals die wordt vastgelegd in het conceptuele schema, is geen beschrijving van de verschillende objecten. Zo zal de beschrijving van de database voor de studentenadministratie geen beschrijving bevatten van de individuele objecten (de verschillende studenten), maar wel van het objecttype student. In plaats van kenmerken of eigenschappen spreken we ook wel van attributen. De attributen van het objecttype student zouden kunnen zijn:

naam
geb. datum
geb. plaats
adres
woonplaats
studienummer
studierichting

Het is nu mogelijk individuele studenten - dus objecten van het objecttype student - in de database vast te leggen door middel van de waarden voor de verschillende attributen.

Voor elk attribuut geldt dat het een vaste betekenis moet hebben en dat het slechts waarden kan aannemen uit een bepaalde verzameling waarden. Zo moet b.v. van het attribuut studierichting van het objecttype student de betekenis eenduidig vaststaan, terwijl het slechts een waarde kan aannemen uit de verzameling van mogelijke studierichtingen. Deze verzameling heet dan het domein waarop het attribuut is gedefinieerd.

In een objecttype kunnen verschillende attributen op hetzelfde domein zijn gedefinieerd. De betekenissen van deze attributen moeten dan wel van elkaar verschillen.

Voorbeeld: het objecttype werknemer met de attributen: werknemer#, naam, afdeling, chef. De attributen werknemer# en chef zijn beide gedefinieerd op het domein van de personeelsnummers, maar hun betekenis is verschillend.

Binnen eenzelfde database kunnen ook in verschillende objecttypen attributen voorkomen die op hetzelfde domein zijn gedefinieerd. Ook in dat geval zal de betekenis verschillend zijn.

Voorbeeld: leverancier: leverancier#, naam, adres, ...
leverantie: leverancier#, art#, prijs, hoeveelheid.
In beide objecttypen is het attribuut leverancier# gedefinieerd op hetzelfde domein. Echter in het objecttype leverancier heeft dit attribuut de betekenis van leverancier# van een potentiële leverancier, terwijl het in het objecttype leverantie het

leverancier# van een leverancier voorstelt die een bepaald artikel in een bepaalde hoeveelheid tegen een bepaalde prijs levert (of geleverd heeft).

Het zal duidelijk zijn dat het vergelijken van de waarden van attributen, b.v. op gelijk, groter of kleiner zijn, alleen maar zinvol kan zijn als de attributen op hetzelfde domein zijn gedefinieerd.

3.3. Relaties tussen objecten

De aard van de relatie tussen de elementen van twee verzamelingen kan verschillend zijn. Laten X en Y twee verzamelingen zijn met elementen x resp. y. We noemen de afbeelding die de elementen van X op Y afbeeldt fx en die van Y op X fy, dus:

$fx : X \rightarrow Y$

$fy : Y \rightarrow X$

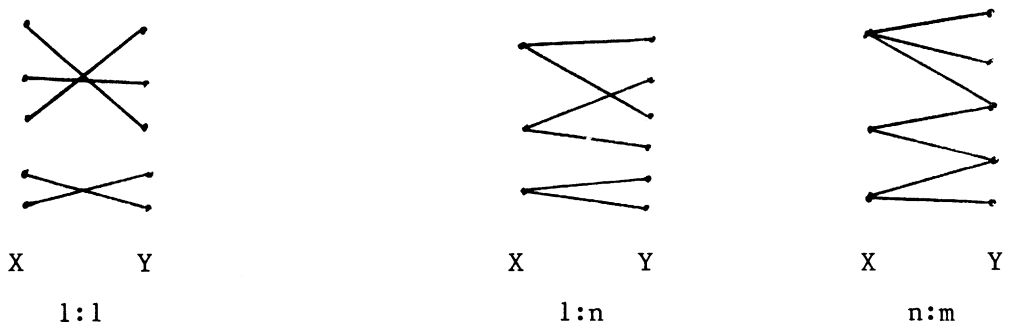
Er geldt:

fx is partieel functioneel als bij elke $x \in X$ hoogstens één $y \in Y$ behoort.
fx is totaal functioneel als bij elke $x \in X$ één $y \in Y$ behoort.

De aard van de relatie kan nu als volgt worden omschreven. Er bestaat tussen de elementen van X en Y een

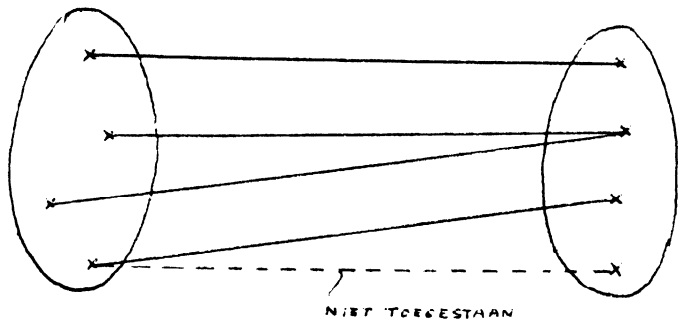
- 1:1 relatie als fx en fy functioneel zijn
- 1:n relatie als of fx of fy functioneel is
- n:m relatie als noch fx noch fy functioneel is.

Een grafische voorstelling hiervan is:



Een relatie tussen objecten van hetzelfde of van verschillend type, kan een 1:1, 1:n of n:m relatie zijn. Zoals in het vervolg zal blijken, hangt de wijze waarop in een database relaties worden vastgelegd af van het gebruikte datamodel.

De relatie tussen objecten en attribuutwaarde is van het type 1:n. Dat wil zeggen dat bij een object één attribuutwaarde van een attribuut hoort, terwijl bij een attribuutwaarde meer dan één object kan horen.



objecttype — attribuut

persoon — geb. datum

In de meeste datamodellen wordt laatstgenoemde relatie op gelijke wijze vastgelegd.

4. OVERZICHT DATAMODELLEN

4.1. Inleiding

Een conceptueel model, vastgelegd in een conceptueel schema, is een beschrijving van de logische structuur van een database. In zo'n model worden de verschillende objecttypen met bijbehorende attributen gedefinieerd.
Zoals eerder is opgemerkt zijn de objecttypen afbeeldingen van dingen in de werkelijkheid, waarvan de niet relevante aspecten zijn weggelaten. Daarom is het conceptuele model slechts een model van de werkelijkheid, dat echter gezien vanuit de beoogde toepassing(en) wel volledig dient te zijn.
Aangezien de werkelijkheid zeer complex is en er bij het ontwerpen van een model vaak verschillende keuzen moeten worden gemaakt, is het mogelijk van een gegeven werkelijkheid verschillende conceptuele modellen te vormen. Hierbij speelt ook de gevolgde ontwerpmethode een belangrijke rol, omdat de mogelijkheden om gegevens te structureren bij de verschillende methoden niet gelijk zijn.

Op grond van de wijze waarop de gegevens zijn gestructureerd, kunnen de conceptuele modellen worden onderscheiden naar type. Een conceptueel model is altijd van een bepaald type datamodel.
In de loop van de tijd zijn er verscheidene datamodellen, ook wel gegevensmodellen genoemd, ontstaan. De drie meest bekende datamodellen zijn op dit moment:

- het hiërarchische model
- het netwerk model
- het relationele model

In de volgende paragrafen worden deze modellen aan de hand van een eenvoudig voorbeeld geintroduceerd.
Voor een uitgebreide behandeling van het semantische model, een model waarin de betekenis van de gegevens een grotere rol speelt dan in de andere modellen, wordt verwezen naar het boek "Database Ontwerp", Ir. J.H. ter Bekke, Stenfert Kroese, 1984 en naar het desbetreffende college.

4.2. Het hiërarchische model

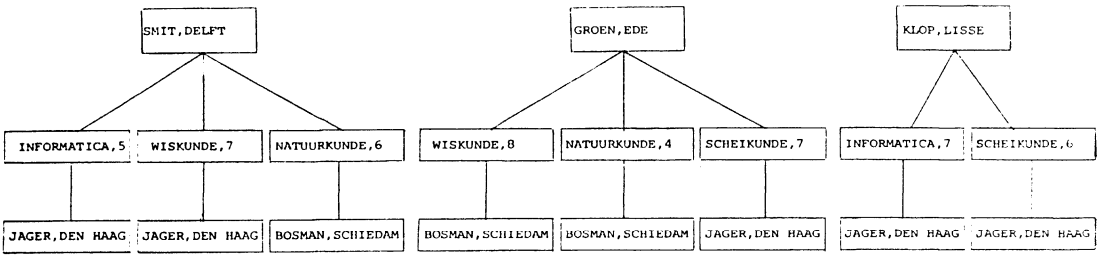
In dit model, dat ontstaan is uit de praktijk van de gegevensverwerking waarbij de opslagmogelijkheden sterk bepalend waren, wordt er van uitgegaan dat er alleen hiërarchische verbanden tussen objecttypen kunnen voorkomen. Bij deze benadering proberen we de samenhang tussen de objecten weer te geven met een boomstructuur. De keuze van de boomstructuur zal i.h.a. worden ingegeven door een bepaalde toepassing, zoals uit het volgende zal blijken. We beschouwen een onderwijssituatie met studenten, vakken en leraren. Van deze "objecttypen" willen we verschillende gegevens registreren. Verder willen we voor het komende semester vastleggen welke studenten bepaalde vakken volgen, bij welke leraren en met welke resultaten. We zien af van een groot aantal details.
Een concreet voorbeeld is:

student Smit, Delft heeft
   informatica van leraar Jager, Den Haag
   wiskunde van leraar Jager, Den Haag
   natuurkunde van leraar Bosman, Schiedam

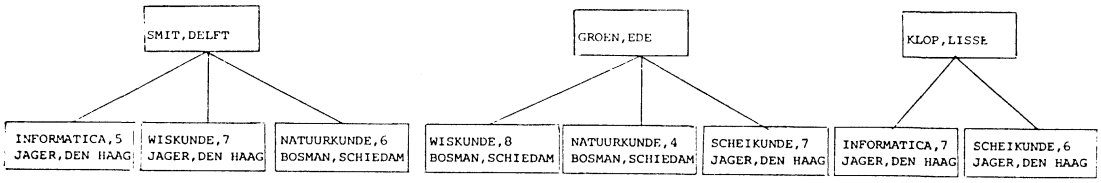
student Groen, Ede heeft
   wiskunde van leraar Bosman, Schiedam
   natuurkunde van leraar Bosman, Schiedam
   scheikunde van leraar Jager, Den Haag

student Klop, Lisse heeft
   informatica van leraar Jager, Den Haag
   scheikunde van leraar Jager, Den Haag

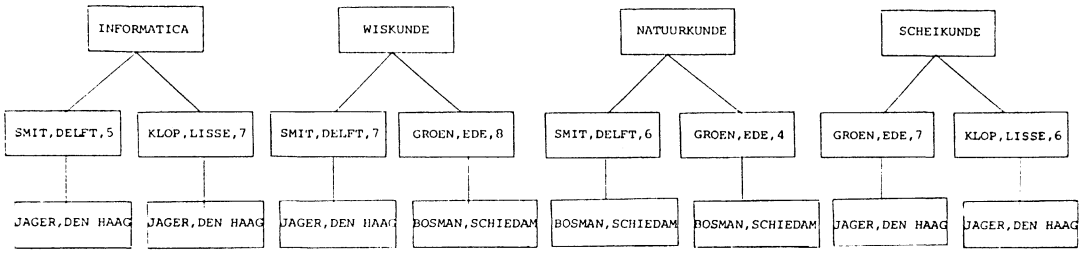
Veronderstellen we nu dat bij veel toepassingen de gegevens per student nodig zullen zijn, dan zou een structuur kunnen worden gekozen waarbij de studentgegevens de toegang tot de andere gegevens bepalen. Een mogelijkheid is:



Daar een student een bepaald vak slechts van één leraar krijgt, kan deze structuur worden vervangen door de volgende:

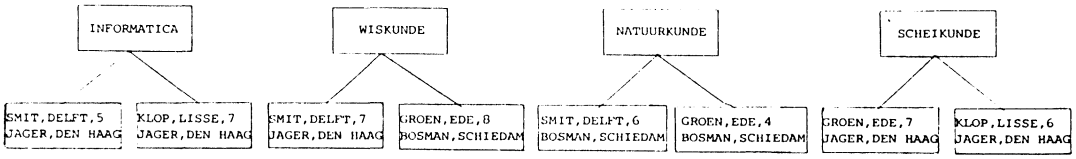


Een andere structuur zou wellicht zijn gekozen als de vakken bij de toepassingen voorop hadden gestaan, bijvoorbeeld:

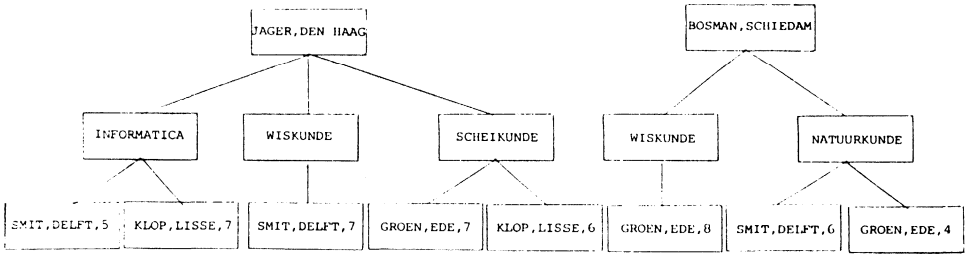


Het cijfer zou in dit geval ook bij de gegevens van de leraar kunenn worden vastgelegd.

Om dezelfde reden als vermeld bij de vorige structuur, kan deze structuur worden vervangen door:

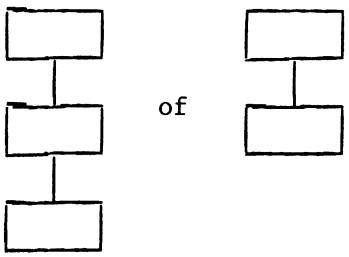


Tenslotte zou ook nog de volgende structuur, met als toegang de gegevens van de leraren, kunnen worden genomen.



Let bij de verschillende structuren op de plaats van het cijfer.

De hier gegeven prentjes zijn voorbeelden van zogenaamde occurrenceplaatjes. De daarbij behorende conceptuele modellen zijn in alle gevallen van de vorm:



waarin alleen de volgorde van de objecttypen (student, vak, leraar) verschillend is. Met  wordt aangegeven dat er een 1:n relatie bestaat

tussen de objecttypen A en B, d.w.z. dat bij één occurrence van het type A n occurrences van het type B behoren.

Is eenmaal voor een bepaalde structuur gekozen, dan zullen voor toepassingen die bij die structuur passen de gegevens op een natuurlijke manier kunnen

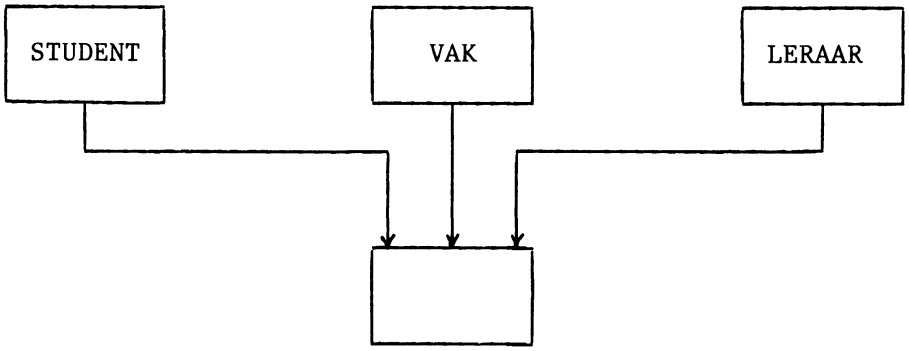
worden benaderd, terwijl voor andere toepassingen nogal gekunstelde programmatuur moet worden ontwikkeld. In sommige implementaties van hiërachische structuren bestaat wel de mogelijkheid om gegevens te benaderen zonder dat dit via de bovengeschikte gegevens gaat. In wezen is er dan sprake van een uitbreiding van de hiërarchische structuur.
Een ander nadeel van de hiërarchische structuur is de vaak voorkomende gegevensovertolligheid.
Het meest bekende hiërarchische DBMS is Information Management System (IMS), een IBM produkt. Het is in 1968 geïntroduceerd en behoort sindsdien tot het meest verspreide databasesysteem in de wereld.

4.3. Het netwerkmodel

Ook dit model is ontstaan uit de praktijk van de gegevensverwerking, waarbij de nadelen van de hiërarchische structuren zeker een rol hebben gespeeld.
Van veel invloed op de ontwikkeling van netwerk georiënteerde databasesystemen zijn de rapporten geweest van de CODASYL - DATA BASE TASK GROUP (eindjaren '60, begin jaren '70). Deze rapporten behelsden voorstellen voor een Data Definition Language en een Data Manipulation Language voor op netwerken gebaseerde databases. Belangrijk in deze rapporten is de scheiding die is aangebracht tussen de logische en de fysieke structuur van de gegevens. Momenteel zijn verschillende Codasyl-achtige database management systemen operationeel. CODASYL is afgeleid van Conference on Data Systems Languages, zie ook 9.1.

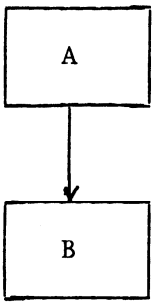
Aangezien het netwerkmodel na het hiërarchische model is ontstaan, heeft het netwerkmodel meer toepassingsmogelijkheden. Dit komt voornamelijk door het feit dat de beperking dat een recordtype -in het hiërarchische model meestal een segment genoemd- kan optreden als ondergeschikte van slechts één ander recordtype, in het netwerkmodel niet meer geldt. Het is daardoor mogelijk een betere afbeelding van de werkelijkheid te krijgen, daar in dit model ook de n:m relatie tussen objecten is weer te geven.

Nemen we weer de onderwijssituatie uit de vorige paragraaf als voorbeeld, dan is een netwerkmodel voor deze toepassing:



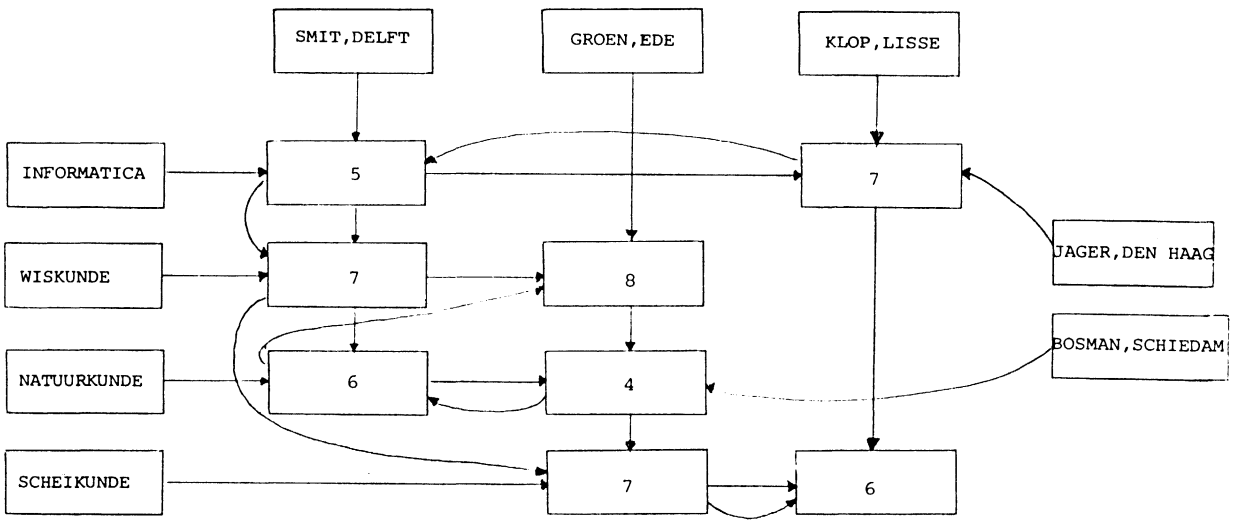
Het ingevoerde recordtype dient om de relaties tussen de andere recordtypen vast te leggen.

Hierin wordt met



aangegeven dat tussen de recordtypen A en B een 1:n relatie bestaat, d.w.z. dat bij één occurrence van het recordtype A n occurrences van het type B behoren.

Het occurrence plaatje bij het voorbeeld is:



In principe zou het ingevoerde recordtype geen gegevens behoeven te bevatten. Worden er wel gegevens in vastgelegd, dan moeten deze betrekking hebben op de combinatie student, vak en leraar. Zo moet het cijfer dat door een student voor een bepaald vak bij een bepaalde leraar is behaald in dit record worden opgenomen.
Elke occurrence van het ingevoerde recordtype komt voor in drie lijsten. De volgorde waarin zij in die lijsten voorkomen is (nog) niet van belang.
We merken op dat de gegevens over studenten, vakken en leraren nu zonder redundantie voorkomen. De samenhang tussen de gegevens is door middel van wijzers (pointers) vastgelegd.
Deze wijzers spelen (helaas) bij de manipulatie een belangrijke rol.

4.4. Het relationele model

Het relationele model is gebaseerd op bestaande wiskundige begrippen, waaronder het begrip relatie. Dit betekent dat dit datamodel, in tegenstelling tot de eerder genoemde modellen niet is afgeleid uit de aanwezige mogelijkheden van gegevensverwerking.

Het verschijnen van een artikel van E.F. Codd (zie 5.1), die als de geestelijke vader van het relationele model kan worden beschouwd, is het begin geweest van veel onderzoek dat op dit terrein in de jaren '70 is gedaan. Verscheidene relationele database management systemen zijn inmiddels operationeel. Momenteel zijn hiervan System R en ORACLE de meest bekende.

In het relationele model wordt geëist dat de occurrences uniek te identificeren zijn. Het is i.h.a. niet handig om hiervoor de eigenschappen (attributen) te gebruiken, daar deze aan verandering onderhevig kunnen zijn (b.v. leraar of student verhuist). Daarom wordt vaak een unieke identificatie toegevoegd. Soms is een dergelijke identificatie al aanwezig, zoals een studienummer voor studenten, een vakcode voor vakken, etc. In dat geval is een toevoeging uiteraard niet nodig.

In deze inleiding op het relationele model willen we volstaan met het geven van de occurrenceplaatjes bij het eerder genoemde voorbeeld.

STUDENT:

stud.id	naam	plaats
S1	Smit	Delft
S2	Groen	Ede
S3	Klop	Lisse

VAK:

vak.id	naam
V1	Informatica
V2	Wiskunde
V3	Natuurkunde
V4	Scheikunde

LERAAR:

ler.id.	naam	plaats
L1	Jager	Den Haag
L2	Bosman	Schiedam

RESULTAAT:

stud.id.	vak.id	ler.id	cijfer
S1	V1	L1	5
S1	V2	L1	7
S1	V3	L2	6
S2	V2	L2	8
S2	V3	L2	4
S2	V4	L1	7
S3	V1	L1	7
S3	V4	L1	6

Een gebruiker kan zich voorstellen dat de gegevens in dergelijke tabellen zijn opgeslagen. In tegenstelling tot het hiërarchische- en het netwerkmodel zijn aan de gegevens nu geen wijzers toegevoegd. De samenhang tussen de gegevens is alleen af te leiden uit het voorkomen van gemeenschappelijke gegevenswaarden.

5. HET RELATIONELE MODEL

5.1. Definitie Relatie

De grondlegger van het relationele model, E.F. Codd, heeft in zijn inmiddels klassiek geworden artikel *) de volgende definitie van een relatie gegeven:

Gegeven zijn de verzamelingen V_1, V_2 ..., V_n die niet verschillend hoeven te zijn.
R is een relatie op de verzamelingen V_1, V_2, ..., V_n indien het een deelverzameling is van het cartesisch produkt $V_1 * V_2 * \ldots * V_n$.
De relatie R is dus een verzameling van elementen $\{v_1, v_2, \ldots, v_n\}$ zodanig dat $v_1 \in V_1$, $v_2 \in V_2$, ..., $v_n \in V_n$.

De elementen $\{v_1, v_2, \ldots, v_n\}$ van R worden tupels (eng.: tuples) genoemd. De verzamelingen V_1, V_2, ..., V_n zijn de domeinen van R. De waarde n is de graad van R.

Een voorbeeld van een relatie van de graad 3 is de relatie WERKNEMER die wordt gedefiniëerd op de domeinen w#, wnaam en salaris.
Deze relatie wordt gerepresenteerd door de onderstaande tabel.

WERKNEMER

w#	wnaam	salaris
123	Jansen	1000
98	Pietersen	1500
57	Smit	900
93	Slager	2000
12	Bosman	3100
106	Winkel	1200
33	Bosland	1750

Elke rij in deze tabel representeert één tupel van de relatie WERKNEMER. Het aantal tupels in deze relatie bedraagt dus 7.

---

*) "A relational model of data for large shared data banks", Comm. of the ACM 13, No. 6, 377-387 (1970)

Aan deze definitie kleeft een belangrijk bezwaar.
De relatie WERKNEMER* die wordt gerepresenteerd door de onderstaande tabel is volgens de definitie verschillend van de eerder genoemde relatie WERKNEMER, ofschoon ze op dezelfde domeinen is gedefinieerd.

WERKNEMER*

w#	wnaam	salaris
123	Jansen	1000
98	Pietersen	1600
57	Smit	1200
12	Bosman	2700
43	Veenhuis	3300
106	Winkel	1200

Het is duidelijk dat de relaties WERKNEMER en WERKNEMER* twee verschillende toestanden van dezelfde database weergeven. Voor de beschrijving van de gegevens in de database is derhalve deze definitie van een relatie niet geschikt, omdat het een beschrijving van de verschillende objecten is en niet van het objecttype (zie paragraaf 3.2).
In de loop van de tijd worden wel van een wisselend aantal objecten veranderlijke gegevens in de database opgenomen, echter wel steeds van dezelfde objecttypen.

In het vervolg van het dictaat zullen we daarom een relatie R op de verzamelingen V_1, V_2, ..., V_n beschouwen als het objecttype R met de eigenschappen V_1, V_2, ..., V_n hetgeen genoteerd wordt als:

$R(V_1, V_2, \ldots, V_n)$

Opgemerkt zij dat de ordening van de eigenschappen niet van belang is. De relaties $R_1(V_1,V_2,V_3)$, $R_2(V_1,V_3,V_2)$ en R_3 (V_3,V_1,V_2) stellen dezelfde relatie voor.

De relatie R is nu niet meer rechtstreeks gedefinieerd op domeinen maar op attributen. Wel is ieder attribuut gedefiniëerd op een domein. Een attribuut

is op deze manier een eigenschap van een objecttype. Ieder object van dit type heeft voor ieder attribuut een waarde uit het overeenkomstige domein van het attribuut.
Attributen kunnen op hetzelfde domein zijn gedefinieerd. Een voorbeeld hiervan is de relatie (zie paragraaf 3.2)

WERKNEMER(w#, wnaam, afdeling, chef)

waarbij de attributen w# en chef beide zijn gedefinieerd op het domein van de personeelsnummers.

5.2. Sleutels

Omdat een tupel een element van een verzameling is, is ieder tupel uniek. Dit betekent dat er tenminste één combinatie van attributen in de relatie moet bestaan die voor ieder tupel een andere waarde heeft, zodat deze combinatie gebruikt kan worden om een tupel te identificeren. Een dergelijke combinatie wordt sleutel (eng.: key) genoemd. Soms bestaat de sleutel uit één attribuut. Het attribuut w# in de relatie WERKNEMER is een sleutel: ieder tupel van WERKNEMER bevat een verschillende waarde voor w# en kan dus worden gebruikt om de tupels van elkaar te onderscheiden.
Het komt ook voor dat alle attributen van de relatie tezamen de sleutel vormen.

Definitie sleutel

Een sleutel S van relatie R is een deelverzameling van attributen van R met de volgende eigenschappen:

1. In elk tupel van R bepaalt de waarde van S eenduidig dat tupel.
2. Geen attribuut van S kan worden weggelaten zonder eigenschap 1 teniet te doen.

In de relatie WERKNEMER (w#, wnaam, geb.dat, salaris) is de attribuutcombinatie w#, wnaam geen sleutel. Niet omdat door een waarde van w#, wnaam een tupel niet eenduidig wordt bepaald, maar omdat door een waarde van w# alleen een tupel al eenduidig wordt bepaald. De attribuutcombinatie wnaam, geb.datum is geen sleutel omdat door een waarde van wnaam, geb.datum een tupel niet eenduidig wordt bepaald. Een database ontwerper die op grond van het feit dat het bedrijf op dit moment geen werknemers met dezelfde naam in dienst heeft die op dezelfde dag zijn geboren, de attribuutcombinatie wnaam,

geb.datum wel als sleutel opvat, zorgt er voor dat het bedrijf geen mensen kan aannemen met dezelfde naam en geboortedatum omdat die niet in de database kunnen worden opgevoerd.

Alle combinaties van attributen in een relatie die aan de definitie van sleutel voldoen heten kandidaat sleutels (eng.: candidate keys). Een attribuut dat deel uitmaakt van tenminste één kandidaat sleutel heet een primair attribuut. De overige attributen heten niet-primaire attributen.

Een van de kandidaat sleutels wordt gekozen als de primaire sleutel (eng.: primary key). Bij het maken van een keuze moeten de volgende regels in acht worden genomen:

1. Geen enkel attribuut van de primaire sleutel kan een z.g. null (≡ niet gedefinieerde) waarde krijgen, daar anders de identificatie van tupels onmogelijk zou zijn. Deze regel staat in de literatuur bekend onder de naam "entity integrity".
2. De attributen van de primaire sleutel moeten niet aan waardeveranderingen onderhevig zijn. Ieder object heeft zijn eigen vaste waarde voor de sleutel; een andere sleutelwaarde betekent dus een ander object.
3. Geen enkel attribuut van de primaire sleutel ontleent zijn waarden aan een domein dat niet door de organisatie zelf wordt bepaald.

Voorbeelden

ad 2. In de relatie AFDELING(afd#, afdnaam, chef) is het attribuut chef wel een kandidaat sleutel maar dit attribuut zal niet worden gekozen als primaire sleutel, omdat de waarde ervan kan veranderen.

ad 3. Indien in de relatie WERKNEMER ook het bankrekeningnummer wordt opgenomen, zal dit attribuut toch niet als primaire sleutel kunnen dienen.

Een primaire sleutel wordt niet alleen gebruikt voor de identificatie van tupels maar tevens om de samenhang tussen tupels uit verschillende relaties weer te geven. Ga in dit verband eens na wat de consequenties kunnen zijn indien sleutelwijziging zou worden toegestaan.

Definitie externe sleutel

Een attribuut van de relatie R_1 is een externe sleutel (eng.: foreign key) indien:

1. het attribuut niet de primaire sleutel van R_1 is en
2. het attribuut op hetzelfde domein is gedefiniëerd als de primaire sleutel van een relatie R_2 (R_1 en R_2 hoeven niet noodzakelijkerwijs verschillend te zijn).

De externe sleutel moet altijd voldoen aan de z.g. referential integrity d.w.z. ðf de waarde van de externe sleutel is null (≡ niet gedefinieerd) ðf gelijk aan de waarde van de primaire sleutel in een tupel van R_2.

Omdat de externe sleutel op hetzelfde domein is gedefinieerd als de primaire sleutel is een vergelijking op waarde mogelijk en zinvol. Tupels met gelijke waarde houden, in een zekere betekenis, verband met elkaar.

Voorbeeld.
Gegeven zijn de relaties

WERKNEMER(w#, wnaam, afd, chef) met sleutel w#

en WERK(proj#, w#, tijdbesteed) met sleutel proj#, w# (proj# is een project-nr.)

In deze relaties zijn twee externe sleutels aanwezig, n.l.

1. w# in WERK;
2. chef in WERKNEMER.

Vanwege de referential integrity moet dus voor iedere waarde van w# in WERK een tupel in WERKNEMER te vinden zijn met dezelfde waarde voor w#.
Zo moet ook voor iedere waarde van chef een tupel in WERKNEMER te vinden zijn met dezelfde waarde voor w#.
Veronderstel dat de database de volgende tupels bevat:

WERKNEMER

w#	wnaam	afd	chef
127	Jansen	X	33
33	Slager	X	null
108	Vis	X	33
84	Bos	Y	27
53	Broer	Y	27
27	Pieter	Y	null
49	Fuik	Y	27

WERK

proj#	w#	tijdbesteed
1	127	80
1	108	136
2	84	54
3	53	212
3	49	4

Het is dan b.v. niet mogelijk

- aan de relatie WERKNEMER een tupel toe te voegen met voor chef de waarde 73.
- aan de relatie WERK een tupel toe te voegen met voor w# de waarde 73.
- uit de relatie WERKNEMER het tupel met voor w# de waarde 33 te verwijderen.
- uit de relatie WERKNEMER het tupel met voor w# de waarde 108 te verwijderen.

5.3. Ontwerpen van sleutels

Beschouwen we de volgende toepassing.

Bij een leverancier worden door klanten bestellingen geplaatst voor artike-

len. Van de klanten worden naam, adres, woonplaats en het nummer van de bankrekening genoteerd. Van de bestellingen de klant, het artikel, de hoeveelheid en het bedrag.

In eerste instantie worden hiervoor de volgende relaties ontworpen:

KLANT (naam, adres, woonplaats, bankrek#)

BESTELLING(naam, adres, woonplaats, art#, hoev, bedrag)

Kandidaatsleutels van KLANT zijn:

1. naam, adres, woonplaats
2. bankrek#

en van BESTELLING:

1. naam, adres, woonplaats, art#.

Geen van deze kandidaatsleutels komt in aanmerking voor primaire sleutel daar de waarden van naam, adres, woonplaats en bankrek# niet door de organisatie zelf worden bepaald en bovendien kunnen veranderen.

Daarom wordt een klantnummer ingevoerd. Hierdoor worden de relaties

KLANT (klant#, naam, adres, woonplaats, bankrek#)

BESTELLING(klant#, art#, hoev, bedrag)

Uit de keuze van de kandidaat sleutels zijn bepaalde gevolgtrekkingen te maken. Daarvan moet steeds worden nagegaan of ze in overeenstemming zijn met de werkelijkheid.

Voorbeelden hiervan zijn:

1. Omdat klant# sleutel is in de relatie KLANT geldt dat iedere klant precies één bankrekening heeft die voor de betalingen van alle bestellingen wordt gebruikt.
   Van een klant die twee bankrekeningen zou hebben, zouden twee tupels met gelijk klant# in de relatie klant moeten voorkomen. Dat kan echter niet omdat klant# sleutel is.
2. Omdat klant#, art# sleutel is in de relatie BESTELLING kan een klant slechts één bestelling van eenzelfde artikel doen. Willen we het een klant mogelijk maken meer dan één bestelling van eenzelfde artikel te kunnen doen, dan moet in de relatie BESTELLING een besteldatum worden opgenomen en worden toegevoegd aan de sleutel.
   Komt de relatie dan wel overeen met de werkelijkheid?

3. Omdat klant# externe sleutel in BESTELLING is, kan een bestelling van een onbekende klant niet worden vastgelegd voordat de gegevens van die klant zijn opgenomen in de relatie KLANT.

5.4. Eerste normaalvorm (1NF)

5.4.1. **Doelstellingen van normaliseren**

Normaliseren is een proces waarbij keer op keer wordt vastgesteld welke attributen niet zijn toegestaan in een relatie om uiteindelijk een database te verkrijgen

1. waarin het mogelijk is iedere relatie met een eenvoudige opslagstructuur vast te leggen;
2. waarop krachtige doch eenvoudige manipulatieopdrachten uitvoerbaar zijn;
3. waarbij geen neveneffecten optreden bij het toevoegen, wijzigen en verwijderen van tupels;
4. waarbij herstructurering van de database bij toevoeging van nieuwe objecttypen eenvoudig is.

5.4.2. **Ongenormaliseerde relatie Bestelling**

We beschouwen de volgende toepassing.
Bij een leverancier worden door klanten bestellingen geplaatst voor artikelen. Op het bestelformulier - dat voorzien is van een uniek bestelnummer - worden gegevens over de klant genoteerd, de besteldatum en per besteld artikel het artikelnummer, de hoeveelheid en de prijs.

Hiervoor wordt de volgende relatie ontworpen:

BESTELLING(b#, bdatum, klant, artikel(art#, hoev, prijs))

Deze relatie bevat op zeker ogenblik de volgende tupels:

b#	bdatum	klant	art#	hoev	prijs	
1	13-10-80	Jansen	10	15	2.00	
			12	5	1.00	één tupel
			17	25	1.25	
2	14-10-80	Smit	8	20	0.50	één tupel
			12	10	0.90	
3	16-10-80	Slager	5	1	5.75	
			6	20	0.25	één tupel
			11	10	3.75	
			12	50	0.70	

Indien we een ongenormaliseerde relatie willen afbeelden op een opslagstructuur kan dit problemen opleveren.
Willen we een tupel als één fysiek record opslaan, dan moeten we de mogelijkheid hebben om met records van variabele lengte te kunnen werken. Soms moet dan wel een limiet worden gesteld aan de lengte van het variabele deel (het aantal artikelen op een bestelling). Indien we niet kunnen beschikken over records met variabele lengte dan moet een tupel in meer dan één fysiek record worden opgeslagen. Deze kunnen dan b.v. in een keten worden opgenomen.

5.4.3. **Definitie 1NF**

Een relatie is in eerste normaalvorm indien alle attributen enkelvoudig (atomair) zijn, d.w.z. zelf geen relatie zijn.

5.4.4. **Décompositie van een relatie naar 1NF**

Gegeven de relatie $R(\underline{A_1}, A_2, A_3, A_4, B(B_1, B_2, B_3))$ waarin A_1 de primaire sleutel is. Deze relatie is niet in 1NF en moet daarom worden gesplitst in twee relaties die wel in 1NF zijn.

De algoritme hiervoor luidt als volgt:
Verwijder uit de relatie het attribuut dat zelf een relatie is en vorm een nieuwe relatie bestaande uit het verwijderde attribuut en de primaire sleutel van de oorspronkelijke relatie. Deze sleutel gaat dan deel uitmaken van de primaire sleutel van de nieuwe relatie. Herhaal indien nodig de algoritme voor de nieuwe relatie.

Toegepast op R geeft dit de relaties:

$R_1(\underline{A_1}, A_2, A_3, A_4)$

$R_2(A_1, B_1, B_2, B_3)$

Het attribuut A_1 maakt deel uit van de primaire sleutel van R_2.

5.4.5. **Relatie Bestelling in 1NF**

De relatie BESTELLING (b#, bdatum, klant, artikel(art#, hoev, prijs)) is niet in 1NF omdat het attribuut artikel zelf een relatie is.
Om relaties in 1NF te krijgen passen we een décompositie overeenkomstig de hiervoor gegeven algoritme toe. Dit geeft de relaties:

BESTELLING (b#, bdatum, klant)

BEST-ARTIKEL(b#, art#, hoev, prijs)

Antwoorden op vragen zoals "worden de artikelen wel of niet steeds voor een vaste prijs geleverd" kunnen op grond van deze relaties niet worden gegeven. Pas na verdere normalisatie kan hierover een uitspraak worden gedaan.

5.4.6. **Relatie Werknemer niet in 1NF**

We beschouwen de volgende toepassing.
Een bedrijf legt over zijn personeelsleden een aantal gegevens vast. De personeelsleden worden van elkaar onderscheiden door een personeelsnummer (p#). Opgenomen worden NAW, geboortedatum van het personeelslid en de namen en geboortedata van zijn/haar kinderen. Het bedrijf houdt vanaf de datum van indiensttreding van ieder personeelslid bij welke funkties vanaf welke datum zijn vervuld en per funktie welk salaris vanaf welke datum werd verdiend.

Hiervoor werd de volgende relatie ontworpen:

WERKNEMER(p#, NAW, geb.dat, kind(geb.dat, knaam),
funktie(fdatum, fnaam, salaris(sdatum, bedrag)))

Deze relatie is niet in 1NF omdat de attributen kind en funktie zelf relaties zijn. Bovendien is binnen funktie het attribuut salaris weer een relatie.
Passen we de eerder beschreven algoritme toe dan krijgen we de relaties:

WERKNEMER (p#, NAW, geb.dat)
KIND (p#, knaam, geb.dat)
FUNKTIE (p#, fdatum, fnaam, salaris(sdatum, bedrag))

De relatie FUNKTIE is niet in 1NF zodat een décompositie van deze relatie noodzakelijk is.
We krijgen dan de relaties:

WERKNEMER (p#, NAW, geb.dat)
KIND (p#, knaam, geb.dat)
FUNKTIE (p#, fdatum, fnaam)
SALARIS (p#, fdatum, sdatum, bedrag)

Opgemerkt zij dat p#, fnaam geen kandidaat sleutel is in de relatie FUNKTIE omdat anders een personeelslid dezelfde funktie niet nog eens kan hebben.
Evenzo is p#, fdatum, bedrag geen kandidaat sleutel in de relatie SALARIS omdat anders een personeelslid hetzelfde salaris niet nog eens kan verdienen.
In de relatie KIND is p#, geb.dat geen kandidaatsleutel omdat van een personeelslid anders geen tweelingen kunnen worden geregistreerd.

Laten we dit resultaat eens aan een nadere beschouwing onderwerpen.

Veronderstel dat personeelslid 127 op 1-8-1978 zijn huidige funktie s.a. heeft gekregen en op 1-10-1980 zijn laatste salarisverhoging tot f. 2.000,-.
De relatie FUNKTIE zal dus het volgende tupel bevatten:

p#	fdatum	fnaam
127	1-8-1978	s.a.

en de relatie SALARIS het tupel:

p#	fdatum	sdatum	bedrag
127	1-8-1978	1-10-1980	2000.-

Op 1-1-1981 krijgt dit personeelslid de nieuwe funktie j.s.o. met behoud van salaris. Eerst op 1-3-1981 ontvangt hij het bij die funktie behorende salaris van fl. 2.300,--.

Aan de relatie FUNKTIE wordt als gevolg hiervan het volgende tupel toegevoegd:

p#	fdatum	fnaam
127	1-8-1978	s.a.
127	1-1-1981	j.s.o.

Op het eerste gezicht lijkt het erop dat als gevolg hiervan aan de relatie SALARIS slechts één tupel hoeft te worden toegevoegd:

p#	fdatum	sdatum	bedrag
127	1-8-1978	1-10-1980	2000.-
127	1-1-1981	1-3-1981	2300.-

Een tupel wordt nu echter eenduidig bepaald door p#, sdatum en dus kan p#, fdatum, sdatum geen sleutel zijn hetgeen wel wordt vereist door de normalisatie algoritme.

Uit de beschrijving van de toepassing valt op te maken dat de relatie SALARIS moet worden gezien als een salaris overzicht per funktie. In deze zin is de datum van een funktieverandering tevens de datum van een salarisverandering.
In dat geval moeten aan de relatie SALARIS twee tupels worden toegevoegd:

p#	fdatum	sdatum	bedrag
127	1-8-1978	1-10-1980	2000.-
127	1-1-1981	1-1-1981	2000.-
127	1-1-1981	1-3-1981	2300.-

Echter ook nu weer wordt een tupel eenduidig bepaald door p#, sdatum, zodat ook deze oplossing moet worden verworpen.

Indien we bij een funktieverandering het salaris dat dan wordt verdiend interpreteren als het aanvangssalaris in die funktie dan worden de volgende tupels aan de relatie SALARIS toegevoegd:

p#	fdatum	sdatum	bedrag
127	1-8-1978	1-10-1980	2000.-
127	1-1-1981	1-10-1980	2000.-
127	1-1-1981	1-3-1981	2300.-

Een tupel wordt nu niet eenduidig bepaald door de attributen p#, sdatum doch door p#, fdatum, sdatum.

Alhoewel deze relaties wel voldoen aan de eisen die 1NF stelt is het geen bevredigende oplossing vanwege de wat curieuze betekenis van de relatie SALARIS. Bovendien ontmoeten we bij het opvoeren van tupels manipulatie problemen, daar eerst de datum van de laatste salarisverandering en het bijbehorende salaris in de database zal moeten worden opgezocht.

Een analyse van de gevolgde methodiek leert ons al snel dat het probleem in wezen wordt veroorzaakt door als uitgangspunt een relatie te nemen waarin

attributen voorkomen die zelf relaties zijn.
In het vervolg zullen we daarom voor het eerste ontwerp slechts relaties met atomaire attributen gebruiken.

5.5. Tweede normaalvorm (2NF)

5.5.1. **Inleiding**

In een relatie in 1NF zijn alle attributen atomair. Het blijkt dat waarden van attributen redundant kunnen zijn opgeslagen en dus tot problemen bij het zoeken in en wijzigen van de relatie kunnen leiden. Aan de hand van het volgende voorbeeld wordt dit geïllustreerd.

5.5.2. **Relatie Scholing niet in 2NF**

Beschouwen we de volgende toepassing.
Een bedrijf legt in een database gegevens vast over zijn personeelsleden en de cursussen die zij hebben gevolgd.
De personeelsleden worden van elkaar onderscheiden door een personeelsnummer(p#). Verder zijn NAW en geboortedatum van ieder personeelslid van belang. Een personeelslid kan in één jaar een of meer cursussen volgen. Een personeelslid volgt een cursus hoogstens éénmaal. De cursussen worden van elkaar onderscheiden door een cursusnummer (c#) en hebben een naam. In een cursus wordt de inhoud van een bepaald boek behandeld. Elk jaar kan een ander boek worden gebruikt. Nadat het personeelslid een cursus heeft gevolgd neemt hij/zij een of meer keren deel aan een examen.

Voor bovenstaande toepassing werd de volgende relatie in 1NF ontworpen:

SCHOLING(p#, NAW, geb.dat, c#, cnaam, cjaar, boek, exdatum, cijfer)

met kandidaat sleutel: p#, c#, exdatum.

Problemen bij het manipuleren met deze relatie zijn o.a.

1. Een personeelslid kan pas in de database worden opgenomen nadat hij zijn eerste examen heeft afgelegd, m.a.w. van personeelsleden die nog geen examen hebben gedaan zijn geen gegevens vastgelegd.

2. Indien een personeelslid aan een examen deelneemt moet niet alleen het examencijfer worden opgevoerd maar ook weer zijn overige gegevens zoals NAW.
3. Indien een personeelslid verhuist dan moet deze wijziging in alle tupels van het betreffende personeelslid worden aangebracht.

De oorzaak van deze problemen moet worden gezocht in ongewenste functionele afhankelijkheden tussen attributen.

5.5.3. **Functionele afhankelijkheid**

Gegeven is de relatie $R(A_1,A_2,A_3,\ldots,A_n)$.
Attribuut A_j in R is functioneel afhankelijk van attribuut A_i in de relatie R indien op ieder moment bij iedere waarde van attrtibuut A_i precies één waarde van attribuut A_j hoort. Of anders gezegd: indien attribuut A_i de waarde van attribuut A_j bepaalt ($i \neq j$).
Deze functionele afhankelijkheid wordt als volgt genoteerd:

$$A_i \rightarrow A_j$$

De waarde van attribuut A_j bij een bepaalde waarde van attribuut A_i kan wel veranderen in de loop van de tijd.
Een voorbeeld hiervan is de functionele afhankelijkheid p# → adres, wpl.
Een personeelslid woont altijd op precies één adres maar dat adres hoeft in de loop van de tijd niet hetzelfde te zijn.

Uiteraard kan deze definitie worden uitgebreid tot een functionele afhankelijkheid van een attribuutcombinatie b.v.

$$A_i,A_j \rightarrow A_k$$

5.5.4. **Volledige functionele afhankelijkheid**

Gegeven de relatie $R(A_1,A_2,A_3,A_4,\ldots,A_n)$.

Het attribuut A_f is volledig functioneel afhankelijk van X, een combinatie van een aantal attributen van R, indien geldt:

1. $X \rightarrow A_f$
2. er kan geen X', een combinatie van een aantal attributen van X met $X' \neq X$, worden gevormd zodanig dat geldt $X' \rightarrow A_f$

Voorbeeld:
Het attribuut A_f is volledig functioneel afhankelijk van de combinatie van attributen A_i, A_j, A_k indien geldt

1. $A_i, A_j, A_k \rightarrow A_f$
2. $A_i, A_j \not\rightarrow A_f$
   $A_i, A_k \not\rightarrow A_f$
   $A_j, A_k \not\rightarrow A_f$

5.5.5. **Definitie 2NF**

Een relatie is in tweede normaalvorm indien de relatie in eerste normaalvorm is en ieder niet-primair attribuut volledig functioneel afhankelijk is van iedere kandidaatsleutel.

5.5.6. **Décompositie van een relatie naar 2NF**

Gegeven de relatie $R(A_1,A_2,A_3,A_4,\ldots,A_n)$ waarvoor geldt:

1. A_1,A_2,A_3 is de enige kandidaatsleutel
2. $A_1,A_2 \rightarrow A_4$.

Deze relatie is niet in 2NF en moet worden gesplitst in twee relaties die wel in 2NF zijn.

De algoritme hiervoor luidt:
Verwijder uit de relatie het niet primaire attribuut dat niet volledig functioneel afhankelijk is van de sleutel en vorm een nieuwe relatie bestaande uit het verwijderde attribuut en de attributen waarvan het wel volledig functioneel afhankelijk is. Deze laatste attributen vormen de primaire sleutel van de nieuwe relatie.

Toegepast op de bovengenoemde relatie geeft dit:

$R_1(\underline{A_1,A_2,A_3},A_5,\ldots,An)$ (zonder A_4) en
$R_2(\underline{A_1,A_2}, A_4)$

5.5.7. **Relatie scholing in 2NF**

In de relatie SCHOLING gelden de volgende functionele afhankelijkheden:

p# → NAW, geb.dat

c# → cnaam

p#,c# → cjaar, boek

c#,cjaar → boek

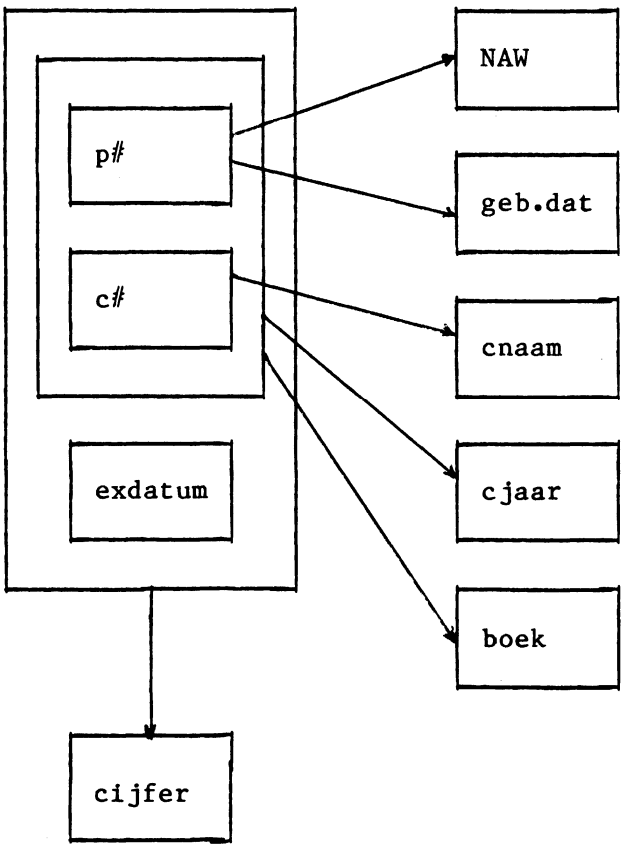
p#,c#,exdatum → cijfer

Op het eerste gezicht zou men kunnen denken dat de functionele afhankelijkheid c# → boek geldt. Echter, dan is de betekenis van boek: het bij de cursus gebruikte boek. Het boek dat gebruikt werd toen een werknemer de cursus volgde, kan alleen maar gelijk zijn aan het huidige boek indien er niets is veranderd. Aangezien deze beperking niet wenselijk is geldt c# → boek dus niet.

Omdat een cursus op het moment dat hij wordt gegeven niet van boek kan veranderen geldt wel de functionele afhankelijkheid:

c#,cjaar → boek

Samengevat krijgen we het volgende schema van de functionele afhankelijkheden waarbij attributen van de sleutel p#,c#,exdatum zijn betrokken:



We constateren dat dus alleen het attribuut cijfer volledig functioneel afhankelijk is van de sleutel p#, c#, exdatum.

Om de relatie in 2NF te krijgen moeten dus alle overige niet primaire attributen uit de relatie worden verwijderd. Dit leidt tot de relaties:

EXAMEN(p#, c#, exdatum, cijfer)
SCHOLING*(p#, c#, NAW, geb.dat, cnaam, cjaar, boek)

De relatie SCHOLING* is niet in 2NF daar NAW, geb.dat en cnaam niet volledig functioneel afhankelijk zijn van de sleutel p#,c#. Na verwijdering van deze attributen uit de relatie SCHOLING* krijgen we de relaties:

EXAMEN(p#, c#, exdatum, cijfer)
SCHOLING**(p#, c#, cjaar, boek)
WERKNEMER(p#, NAW, geb.dat)
CURSUS (c#, cnaam)

Opgemerkt wordt dat de relatie SCHOLING** in 2NF is ook al geldt de functionele afhankelijkheid c#, cjaar → boek.
Ga na dat de in paragraaf 5.5.2. geschetste problemen bij het manipuleren zich bij deze relatie niet voordoen.

Op grond van de functionele afhankelijkheden die al dan niet moeten gelden kan van bepaalde uitspraken worden vastgesteld of ze al dan niet juist zijn. Voorbeelden hiervan zijn:

1. Een personeelslid kan een cursus slechts éénmaal volgen.
   Antwoord:
   Deze bewering is juist omdat de functionele afhankelijkheid p#,c# → cjaar geldt.
2. Een personeelslid kan aan hoogstens één examen van een cursus deelnemen.
   Antwoord:
   Veronderstel dat de bewering juist is, dan zou de functionele afhankelijkheid p#,c# → cijfer moeten gelden. Echter, dan zou de relatie EXAMEN niet in 2NF zijn. Vanwege tegenspraak is de bewering dus onjuist.

5.5.8. **Oefenvraagstuk Niet betaalde fakturen**

Een bedrijf legt in een database gegevens vast over de nog niet betaalde fakturen van klanten. De klanten bestellen bij het bedrijf op een bestelling één of meer artikelen tegen een bepaalde prijs. Een bestelling wordt altijd in zijn geheel afgeleverd en voorzien van een gedateerde faktuur.
De fakturen worden van elkaar onderscheiden door een faktuurnummer. Op de faktuur worden bestelnummer (uniek), besteldatum, het door de klant te betalen bedrag, en per artikel het aantal en de prijs vermeld.

Hiervoor ontwerpt men de volgende relatie:

FAKTUUR(klant, fakt#, b#, bdatum, fdatum, art, aantal, prijs)
met als sleutels de attribuutcombinaties: 1. fakt#, art.
2. b#, art

Herleid deze relatie tot relaties in 2NF.

Opgemerkt wordt dat het totaal te betalen bedrag niet in de database wordt opgenomen daar er anders redundantie zou ontstaan, omdat dit bedrag is af te leiden uit de geleverde artikelen.
In principe zou het gezien de aard van de toepassing niet nodig zijn om de geleverde artikelen in de database op te nemen. Er zou kunnen worden volstaan met het totaal te betalen bedrag.

5.6. Derde normaalvorm (3NF)

5.6.1. **Inleiding**

In een relatie in 2NF zijn de niet-primaire attributen volledig functioneel afhankelijk van alle kandidaatsleutels. Het blijkt echter dat functionele afhankelijkheden tussen niet-primaire attributen ook tot redundantie en dus tot problemen bij het zoeken in en wijzigen van relaties kunnen leiden. Aan de hand van het volgende voorbeeld wordt dit geïllustreerd.

5.6.2. **Relatie WERKNEMER niet in 3NF**

Beschouwen we de volgende toepassing.
Een ingenieursbureau legt in een database gegevens vast over zijn werknemers.
De werknemers worden van elkaar onderscheiden door een intern nummer (w#). Verder zijn NAW en geboortedatum bekend. Iedere werknemer heeft één funktie en werkt op één afdeling. Iedere afdeling heeft één chef en verricht werkzaamheden voor één externe klant.

Hiervoor wordt de volgende relatie ontworpen.

WERKNEMER(<u>w#</u>, NAW, geb.dat, funktie, afd, chef, klant)

met als enige sleutel w#.

Deze relatie is in 2NF daar ieder niet-primair attribuut volledig functioneel afhankelijk is van iedere kandidaatsleutel.

Toch zijn er de volgende manipulatieproblemen.

1. Een nieuwe afdeling kan pas worden opgevoerd nadat er een werknemer op is komen te werken, m.a.w. van afdelingen waarop nog geen mensen werken kunnen geen gegevens worden opgenomen;
2. Indien het bedrijf een nieuwe werknemer krijgt dan moeten ook de gegevens van de afdeling weer worden opgevoerd;
3. Indien een afdeling een nieuwe chef krijgt dan moet deze wijziging bij alle werknemers van die afdeling worden aangebracht.

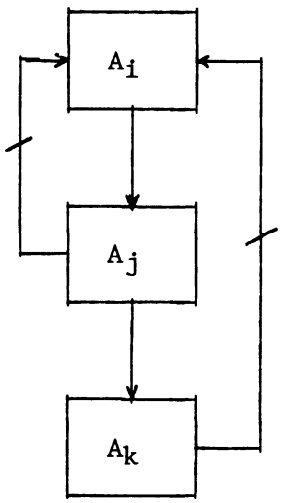
Deze problemen worden veroorzaakt door functionele afhankelijkheden tussen niet-primaire attributen.

5.6.3. **Transitieve functionele afhankelijkheid**

Gegeven is de relatie $R(A_1,A_2,A_3,\ldots,A_n)$.
Een attribuut A_k in R is transitief functioneel afhankelijk van het attribuut A_i indien er een attribuut A_j te vinden is, zodanig dat geldt:

1. $A_i \rightarrow A_j$
2. $A_j \rightarrow A_k$ en dus $A_i \rightarrow A_k$
3. $A_k \not\rightarrow A_i$
4. $A_j \not\rightarrow A_i$

Schematisch weergegeven levert dit het volgende beeld:



5.6.4. **Definitie 3NF**

Een relatie is in derde normaalvorm indien de relatie in tweede normaalvorm is en geen enkel niet-primair attribuut transitief functioneel afhankelijk is van een kandidaatsleutel.

5.6.5. **Décompositie van een relatie naar 3NF**

Gegeven de relatie $R(A_1,A_2,A_3,A_4,\ldots,A_n)$ waarvoor geldt:
1. A_1 is de enige kandidaatsleutel;
2. A_3 is via A_2 transitief afhankelijk van A_1.

Deze relatie is niet in 3NF en en moet daarom worden gesplitst in twee relaties die wel in 3NF zijn.
De algoritme hiervoor luidt:
Verwijder uit de relatie het attribuut dat transitief afhankelijk is van een sleutel en vorm een nieuwe relatie bestaande uit het verwijderde attribuut en het attribuut waarlangs de transitieve afhankelijkheid loopt. Dit laatste attribuut vormt de primaire sleutel van de nieuwe relatie.

Toegepast op bovengenoemde relatie geeft dit de relaties:

$R_1(\underline{A_1}, A_2, A_4, \ldots, A_n)$ (zonder A_3)

$R_2(\underline{A_2}, A_3)$

5.6.6. **Relatie WERKNEMER in 3NF**

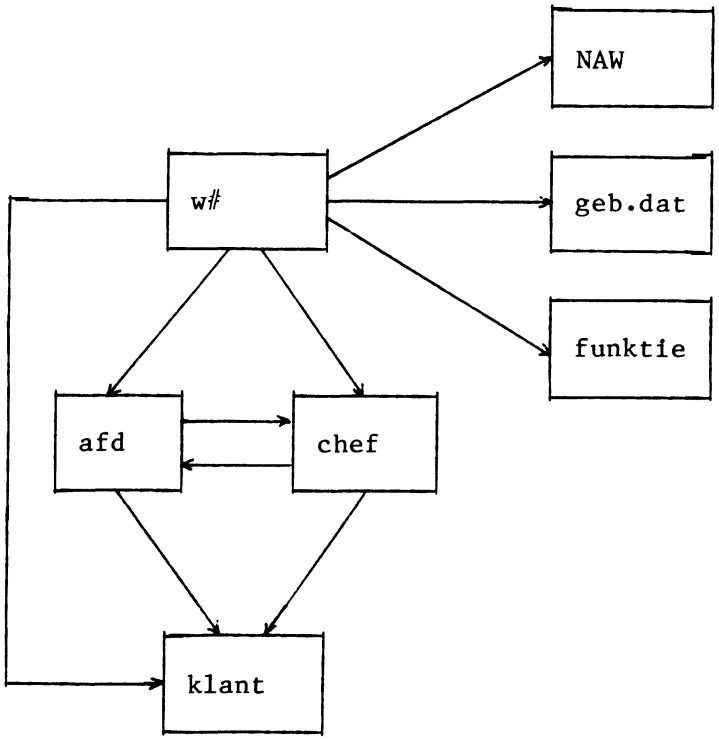
In de relatie WERKNEMER gelden de volgende functionele afhankelijkheden:

w# → NAW, geb.dat, funktie, afd, chef, klant

afd → chef, klant

chef → afd, klant.

Schematisch weergegeven:



Hieruit leiden we af dat de volgende vier transitieve afhankelijkheden gelden:

1. w# → afd → chef
2. w# → chef → afd
3. w# → afd → klant
4. w# → chef → klant

Deze vier transitieve afhankelijkheden moeten een voor een worden verwijderd door décompositie. De volgorde waarin ze worden verwijderd kan willekeurig worden gekozen. Een bepaalde keuze kan echter wel eens leiden tot een onbevredigend resultaat.
De keuze valt op de transitieve afhankelijkheid w# → afd → klant. Na verwijdering ontstaan de volgende relaties:

WERKNEMER*(w#, NAW, geb.dat, funktie, afd, chef)
OPDRACHTGEVER(afd, klant)

Helaas bevat de relatie WERKNEMER* nog de volgende twee transitieve afhankelijkheden:
1. w# → afd → chef
2. w# → chef → afd.
De keuze valt nu op verwijdering van de transitieve afhankelijkheid w# → chef → afd. Er ontstaan dan de relaties:

OPDRACHTGEVER(afd, klant)
WERKNEMER**(w#, NAW, geb.dat, funktie, chef)
HOOFD(chef, afd)

In de relatie HOOFD zijn de attributen chef en afd beide kandidaat sleutel. Omdat de relaties OPDRACHTGEVER en HOOFD beide dezelfde kandidaat sleutel hebben kunnen de relaties worden samengevoegd.
Het resultaat van de decompositie zijn dus de volgende twee relaties in 3NF:

WERKNEMER**(w#, NAW, geb.dat, funktie, chef)
HOOFD* (chef, afd, klant).

In de relatie HOOFD* moet de keuze voor de primaire sleutel vallen op het attribuut chef. Als de keuze op het attribuut afd zou vallen, dan zou het attribuut chef in WERKNEMER** geen externe sleutel zijn wat inhoudt dat de referential integrity niet wordt gegarandeerd. In dit geval zou dit betekenen dat in de database niet bij iedere werknemer de eigenschappen van zijn chef (afd, klant) worden gevonden.

Alhoewel de relaties nu in 3NF zijn, is het resultaat toch niet bevredigend.

Als de werknemers op een afdeling een nieuwe chef krijgen dan moet deze wijziging bij iedere werknemer worden aangebracht. Dit komt omdat de betekenis van het attribuut chef in de relatie WERKNEMER** niet de chef van de afdeling is maar de chef van de werknemer. In de relatie HOOFD* worden dan ook de eigenschappen van een chef beschreven en niet van een afdeling!

Door in een andere volgorde de transitieve afhankelijkheden te verwijderen wordt een ander resultaat verkregen. Eerst wordt w# → afd → klant verwijderd. Dit geeft:

WERKNEMER*(w#, NAW, geb.dat, funktie, afd, chef)
OPDRACHTGEVER(afd, klant)

Daarna wordt w# → afd → chef verwijderd. Dit leidt tot:

WERKNEMER**(w#, NAW, geb.dat, funktie, afd)
AFDELING(afd, chef, klant)

Indien nu de werknemers op een afdeling een nieuwe chef krijgen behoeft slechts één tupel van de relatie AFDELING te worden gewijzigd.

Ook nu kan van bepaalde beweringen worden vastgesteld of ze juist zijn. Voorbeelden hiervan zijn:

1. Een werknemer kan werkzaamheden verrichten voor meer dan één externe klant.

   Antwoord:

   Deze bewering is onwaar daar uit de functionele afhankelijkheden w# → afd en afd → klant volgt dat w# → klant en dus een werknemer werkzaamheden voor slechts één klant verricht.

2. Eenzelfde funktie wordt niet op twee of meer afdelingen vervuld.

   Antwoord:

   Veronderstel dat de bewering waar is. Dan zou de functionele afhankelijkheid funktie → afd moeten gelden, maar dan zou afd transitief afhankelijk zijn van w# (via funktie) en dan zou de relatie WERKNEMER** niet in 3NF zijn. Vanwege tegenspraak is de bewering dus onwaar.

3. Een klant kan aan twee of meer afdelingen opdrachten verstrekken voor het verrichten van werkzaamheden.

   Antwoord:

   Veronderstel dat deze bewering onwaar is. Dan zou de functionele afhankelijkheid klant → afd en dus ook klant → chef moeten gelden. In dat geval zou klant kandidaat sleutel zijn in de relatie AFDELING. Echter daarin is

klant niet als sleutel gedefinieerd. Vanwege tegenspraak is de bewering dus waar.

5.6.7. **Oefenvraagstuk Touroperator**

Een touroperator maakt voor het verzorgen van busreizen naar de verschillende wintersportgebieden in Europa gebruik van een database.
De touroperator beschikt over een groot aantal bussen die van elkaar worden onderscheiden door een busnummer.
Voor iedere bus wordt een rittenschema gemaakt. Een rit voert van of naar de hotels in een gebied. Een bus maakt op een dag hoogstens een heen- of een terugreis. Het is niet ongebruikelijk dat in drukke perioden op hetzelfde gebied al dan niet op dezelfde dag meer dan één bus wordt ingezet. Het aantal personen dat met een bus kan worden vervoerd is niet voor alle bussen hetzelfde.

De kosten van een busreis worden berekend op basis van de prijs per persoon voor het vervoer van en naar het gebied. Deze prijs verandert gedurende het seizoen niet.

Hiervoor wordt de volgende relatie ontworpen:

rit(<u>bus#, datum</u>, gebied, richting, aantal pers, prijs)

waarin richting de waarde heen of terug heeft.

Ontwerp op grond hiervan een relatiemodel in 3NF voor deze database.

5.7. <u>Boyce-Codd normaalvorm</u> (BCNF)

5.7.1. **Inleiding**

In een relatie in 3NF zijn de niet-primaire attributen volledig functioneel afhankelijk van elke kandidaat sleutel en niet functioneel afhankelijk van elkaar. Het blijkt echter dat functionele afhankelijkheden tussen primaire

attributen eveneens tot problemen kunnen leiden. Aan de hand van het volgende voorbeeld wordt dit geïllustreerd.

5.7.2. **Relatie Advies niet in BCNF**

Een verzekeringsmaatschappij heeft experts in dienst om klanten te adviseren ten aanzien van mogelijk te leveren diensten, zoals verschillende vormen van verzekeringen, lijfrentepolissen, beleggingen, hypotheken, e.d.
Klanten kunnen op meer dan één gebied advies krijgen. Een klant krijgt echter op een bepaald gebied advies van slechts één expert.
De experts zijn gespecialiseerd op één gebied, terwijl op een gebied een aantal experts kunnen adviseren.

De relatie die hiervoor wordt ontworpen is:

ADVIES (klant, gebied, expert)

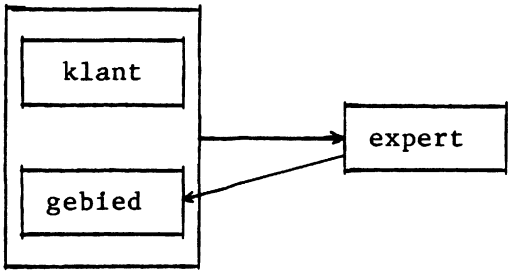
met kandidaat sleutels 1. klant, gebied
2. klant, expert.

Deze relatie is in 3NF alleen al vanwege het feit dat alle attributen primaire attributen zijn.

Problemen bij het manipuleren met deze relatie zijn o.a.

1. Experts zijn pas in de database opgenomen indien ze tenminste één klant adviseren, m.a.w. van experts die op dit moment geen enkele klant (meer) adviseren is het specialisme niet in de database vastgelegd.
2. Indien een expert uitvalt en zijn werkzaamheden overgenomen worden door een andere expert, dan moet deze wijziging bij al zijn klanten worden aangebracht.

Deze problemen worden veroorzaakt door functionele afhankelijkheden tussen primaire attributen:



We zeggen dat deze relatie niet in BCNF is.

5.7.3. **Definitie BCNF**

De relatie $R(A_1, A_2, A_3, ..., A_n)$ is in BCNF indien geldt
1. de relatie R is in 1NF;
2. als geldt $A_i \rightarrow A_j$ $(i \neq j)$ dan moet voor elke k (k = 1,2,3,..,n) gelden $A_i \rightarrow A_k$.

In Date *) wordt een andere definitie gegeven die echter wel gelijkwaardig is.
In de relatie $R(A_1, A_2, A_3, ..., A_n)$ is A_i een determinant indien er een k bestaat $(k \neq i)$ zodat geldt: $A_i \rightarrow A_k$.
De relatie R is in BCNF indien:
1. de relatie in 1NF is en
2. iedere determinant een kandidaat sleutel is.

In bovengenoemde definities kan A_i uiteraard ook een samengesteld attribuut zijn.

*) C.J. Date, An Introduction to Database Systems (3rd. ed.) Addison Wesley.

5.7.4. **Décompositie van een relatie naar BCNF**

Gegeven de relatie $R(\underline{A_1}, A_2, A_3, ..., A_j, ..., A_n)$ waarvoor geldt:
1. $A_i \rightarrow A_j$ $(i \neq j)$
2. er is een k waarvoor geldt $A_i \not\rightarrow A_k$.
Deze relatie is dus niet in BCNF en moet daarom worden gesplitst in twee

relaties die wel in BCNF zijn.

De algoritme hiervoor luidt:
Verwijder uit de relatie het attribuut dat functioneel afhankelijk is van een attribuut dat geen kandidaatsleutel is en vorm een nieuwe relatie bestaande uit het verwijderde attribuut en het attribuut waarvan het functioneel afhankelijk is. Dit laatste attribuut is de primaire sleutel van de nieuwe relatie.

Bovengenoemde relatie R wordt dus gesplitst in:

$R_1(\underline{A_1}, A_2, A_3, \ldots, A_n)$ (zonder A_j)
en
$R_2(\underline{A_i}, A_j)$

5.7.5. **Relatie Advies in BCNF**

De relatie ADVIES (klant, gebied, expert) is vanwege

expert → gebied
expert ↛ klant

niet in BCNF. Om relaties in BCNF te krijgen vindt de volgende décompositie plaats:

ADVIES* (<u>klant, expert</u>)
SPECIALIST (<u>expert</u>, gebied).

Nadere beschouwing van dit resultaat leert ons:

1. Er wordt meer vastgelegd dan met de oorspronkelijke relatie ADVIES, n.l. welke experts de verzekeringsmaatschappij in dienst heeft en wat hun specialisme is. Daarnaast is ook nu weer vastgelegd welke expert aan welke klant op welk gebied adviseert.
2. Het is nu mogelijk een tupel van ADVIES* op te voeren waardoor in de database wordt vastgelegd dat een klant op één gebied van twee experts advies krijgt. Stel de database bevat de volgende tupels:

ADVIES*

klant	expert
K1	E1
K2	E3
K2	E2
K3	E1

en van

SPECIALIST

expert	gebied
E1	hypotheken
E2	lijfrente
E3	hypotheken

Indien nu tupel (K3, E3) wordt opgevoerd, zal deze opvoering door het DBMS niet worden geweigerd. Hierna is dan in de database vastgelegd dat klant K3 op het gebied hypotheken van expert E1 en E3 advies krijgt.

De oorspronkelijke relatie ADVIES zou de volgende tupels bevatten:

ADVIES

klant	expert	gebied
K1	E1	hypotheken
K2	E3	hypotheken
K2	E2	lijfrente
K3	E1	hypotheken

In dit geval zou de opvoering van tupel (K3, E3, hypotheken) wel door het

DBMS worden geweigerd, omdat de sleutel (K3, hypotheken) al in de database voorkomt.

3. Ook na splitsing moet, indien de werkzaamheden van een expert worden overgenomen door een andere expert, deze wijziging bij al zijn klanten worden aangebracht.

Deze beschouwing werpt de vraag op of de splitsing van ADVIES in ADVIES* en SPECIALIST eigenlijk wel wenselijk is. Daarvoor onderscheiden we twee situaties:

1. Naast gegevens over het adviseren worden ook gegevens van experts vastgelegd. Een splitsing is dan onvermijdelijk. Programmatisch moet dan worden voorkomen dat een klant op een gebied van meer dan één expert advies krijgt.

2. Alleen gegevens over het adviseren worden vastgelegd. Omdat de splitsing van de relatie gebeurde op grond van de geschetste problemen (par. 5.7.2) en deze kunnen worden weerlegd, is de splitsing niet wenselijk.
   Het eerst probleem - indien een expert niet adviseert zijn er geen gegevens over hem vastgelegd - vervalt, daar we in die gegevens niet geïnteresseerd zijn.
   Het tweede probleem - de werkzaamheden van een expert worden overgenomen door een andere expert - is een suggestieve manier van voorstellen van de werkelijkheid. Wanneer een expert uitvalt zal iedere klant die hij adviseert een andere expert krijgen toegewezen. Omdat dit niet voor iedere klant dezelfde expert behoeft te zijn, zal deze wijziging bij iedere klant afzonderlijk moeten worden aangebracht.

5.7.6. **Relatie Les**

In het eerder genoemde boek van Date wordt in het hoofdstuk "Further Normalization" aan de hand van het volgende voorbeeld de noodzaak van BCNF aangetoond.

Een onderwijsinstelling legt in een database vast welke studenten welk vak bij welke docent volgen. Een student volgt een bepaald vak bij één docent.

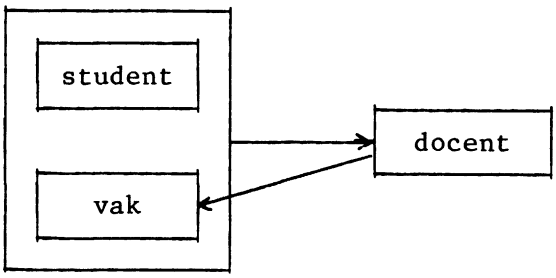
Een vak kan door verschillende docenten worden gegeven, maar een docent kan hoogstens één vak geven.

Hiervoor wordt de volgende relatie ontworpen:

LES (student, vak, docent)

met kandidaat sleutels:
1. student, vak
2. student, docent.

Deze relatie is in 3NF maar niet in BCNF omdat de volgende functionele afhankelijkheid geldt:



Décompositie van LES levert:

LES* (<u>student, docent</u>)

en

LERAAR(<u>docent</u>, vak)

De argumentatie om tot een splitsing over te gaan is het vermijden van de problemen:

1. Docenten zijn pas in de database opgenomen indien ze lesgeven m.a.w. van docenten die op dit moment geen les geven is niet bekend welk vak zij kunnen doceren.
2. Indien een docent uitvalt en zijn les wordt overgenomen door een andere docent, dan moet deze wijziging bij alle studenten van deze les worden aangebracht.

Tegen décompositie van de relatie LES zijn, net zoals bij het voorbeeld ADVIES, de volgende bezwaren in te brengen:

1. Er wordt meer vastgelegd dan met de oorspronkelijke relatie LES n.l. welk vak de docenten kunnen geven;
2. Het is mogelijk een tupel van LES* op te voeren waardoor in de database wordt vastgelegd dat een student hetzelfde vak volgt bij twee docenten;
3. Ook na splitsing moet indien het lesgeven door een andere docent wordt overgenomen, deze wijziging bij alle studenten die het vak volgen worden aangebracht.

Afgezien van deze bezwaren is er nog een ander bezwaar, n.l. dat de voorwaarde dat een docent slechts één vak kan doceren, niet erg realistisch maar wel essentieel is voor het voorbeeld.

Als deze voorwaarde niet zou worden gehanteerd dan zou gelden docent $\not\rightarrow$ vak. De relatie LES(student, vak, docent) heeft dan slechts één sleutel: student, vak. Omdat deze relatie in BCNF is, vindt er geen splitsing plaats. Toch kent deze relatie problemen:
1. Docenten zijn pas in de database opgenomen indien ze les geven.
2. Indien een docent uitvalt, moet deze wijziging bij alle studenten die deze les volgen worden aangebracht.

Blijkbaar zijn relaties in BCNF geen probleemloze relaties. Dit heeft geleid tot een vierde en zelfs een vijfde normaalvorm. Omdat deze normaalvormen meer vragen oproepen dan beantwoorden, beperken we ons in het college tot BCNF.

5.7.7. **Relatie Uitslaglijst**

Met het volgende voorbeeld laten we zien dat in een relatie met functionele afhankelijkheden tussen primaire attributen niet altijd sprake hoeft te zijn van redundantie en dus van problemen bij het wijzigen van of zoeken in de relatie.

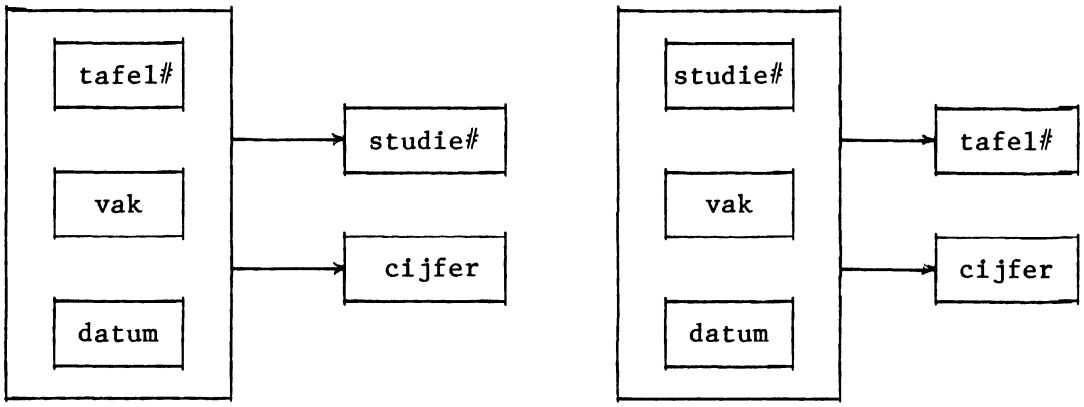
Een onderwijsinstelling legt de uitslagen van tentamens in een database vast. Op de uitslagen wordt per deelnemer tafelnummer, studienummer, vak, datum en cijfer vermeld.

De relatie die hiervoor wordt ontworpen luidt:

UITSLAG (tafel#, studie#, vak, datum, cijfer)

met kandidaat sleutels: 1. tafel#, vak, datum
2. studie#, vak, datum.

Tussen de attributen bestaan de volgende functionele afhankelijkheden:



Deze relatie is in BCNF en geeft geen problemen.

Het verschil met de vorige relaties is dat hier alle attributen (ook de primaire) volledig functioneel afhankelijk zijn van de sleutels waartoe het attribuut niet behoort. De relatie ADVIES voldoet hier niet aan omdat gebied niet volledig functioneel afhankelijk is van klant, expert.

5.7.8. **Relatie Scholing in BCNF**

Voor het bedrijf dat de scholing van zijn werknemers in een database wil vastleggen worden de volgende relaties in 2NF ontworpen (zie paragraaf 5.5.7.):

WERKNEMER(p#, NAW, geb.dat)
SCHOLING(p#, c#, cjaar, boek)
EXAMEN(p#, c#, exdatum, cijfer)
CURSUS(c#, cnaam)

Deze relaties zijn vanwege het ontbreken van transitieve afhankelijkheden tevens in 3NF.
Ze zijn echter niet in BCNF vanwege
1. c#, cjaar → boek
2. c#, cjaar ↛ p#

Na décompositie van de relatie SCHOLING ontstaan de volgende relaties die wel in BCNF zijn:

WERKNEMER(p#, NAW, geb.dat)
SCHOLING(p#, c#, cjaar)
CBOEK(c#, cjaar, boek)
EXAMEN(p#, c#, exdatum, cijfer)
CURSUS(c#, cnaam)

Het zal duidelijk zijn dat in een praktijksituatie de relaties CBOEK en CURSUS zullen worden samengevoegd tot de relatie CURSUS*(c#,cjaar, cnaam, boek). Deze relatie is dan niet in 2NF!

5.7.9. **Oefenvraagstuk: Afleveren bestellingen**

Een bedrijf heeft klanten die artikelen bestellen. Een klant heeft een k # (uniek), naam, adres, woonplaats, postcode, een kredietlimiet, een debetbedrag (nog niet betaalde rekeningen) en één of meer afleveradressen (= straatnaam, nummer, plaatsnaam). De afleveradressen worden over de klanten heen van elkaar onderscheiden door het afadr#. Op een afleveradres kunnen geen goederen van verschillende klanten worden bezorgd.

Een klant plaatst op één dag hoogstens één bestelling. Hiervoor wordt aanvankelijk de volgende relatie ontworpen:

AFLEVERING(b#, datum, klant, NAW, krediet, debet, afadr#, straatnaam, nummer, plaatsnaam)

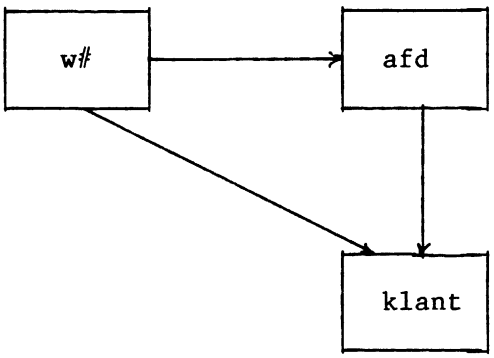
Maak een relatiemodel in BCNF voor deze database.

5.8. Goede en slechte décomposities

We beschouwen de relatie

WERKNEMER (w#, afd, klant)

met de volgende functionele afhankelijkheden:



Omdat de relatie niet in 3NF is wordt hij gesplitst in:

WERKNEMER (w#, afd) en

AFDELING (afd, klant)

Hieruit is af te leiden:

1. op welke afdeling een werknemer werkt daar w# → afd;
2. voor welke klant een afdeling werkt daar afd → klant;
3. voor welke klant een werknemer werkt daar uit w# → afd en afd → klant volgt w# → klant.

Andere mogelijke décomposities zijn:

1. WERKNEMER (w#, klant)
   AFDELING (afd, klant)
2. WERKNEMER (w#, afd)
   WERK KLANT (w#, klant)

ad 1. Uit deze relaties is af te leiden voor welke klant een afdeling en een werknemer werkt. Echter de informatie op welke afdeling een werknemer werkt gaat verloren omdat

   a. w# → afd niet is af te leiden uit de functionele afhankelijkheden w# → klant en afd → klant;
   b. de afdeling van een werknemer niet m.b.v. een zoekproces in de database gevonden kan worden.

ad 2. Uit deze relaties is af te leiden op welke afdeling en voor welke klant een werknemer werkt. De informatie voor welke klant een afdeling werkt is niet rechtstreeks afleidbaar, maar gaat toch niet verloren daar met het volgende zoekproces in de database de klant van een afdeling kan worden gevonden.

Zoek in de relatie WERKNEMER een tupel waarvan de waarde van het attribuut afd gelijk is aan de betreffende afdeling. De waarde van w# in dit tupel representeert een werknemer van de afdeling. Zoek daarna in de relatie WERK KLANT het tupel waarvan de waarde van het attribuut w# gelijk is aan de zoëven gevonden waarde. De waarde van klant in dit tupel representeert de klant van de werknemer op de betreffende afdeling en dus tevens de klant van de afdeling.

Het wezenlijke probleem is dat er programmatisch steeds voor moet worden gezord dat alle werknemers van een afdeling voor dezelfde klant werken omdat anders niet aan de functionele afhankelijkheid afd→klant wordt voldaan.

Décompositie van een relatie heeft tot gevolg dat één van de functionele afhankelijkheden niet meer binnen de relatie zelf wordt afgedwongen (intrarelational constraint) maar over de relaties heen (interrelational contraint). Passen we niet de goede décompositie toe, dan wordt aan de interrelational constraint niet automatisch voldaan en zijn de relaties niet onafhankelijk van elkaar te wijzigen.

Keren we terug tot het voorbeeld dan zien we dat de interrelational constraint bij de décompositie:

1. WERKNEMER (w#, afd)
   AFDELING (afd, klant)
   de functionele afhankelijkheid w# → klant is, waaraan automatisch wordt voldaan door de twee intrarelational constraints w# → afd en afd→klant;
2. WERKNEMER (w#, klant)
   AFDELING (afd, klant)
   de functionele afhankelijkheid w# → afd is, waaraan niet wordt voldaan;

3. WERKNEMER (w#, afd)
   WERK KLANT (w#, klant)
   de functionele afhankelijkheid afd → klant is, waaraan niet automatisch wordt voldaan door intrarelational constraints maar programmatisch moet worden afgedwongen.

Rissanen heeft aangetoond dat na splitsing van R in R1 en R2, R1 en R2 onafhankelijk van elkaar gewijzigd kunnen worden dan en slechts dan indien
1. het gemeenschappelijk attribuut een kandidaat sleutel in R1 of R2 is en
2. elke functionele afhankelijkheid in R logisch afleidbaar is uit de functionele afhankelijkheden in R1 en R2.

Beschouwen we nu nogmaals de relatie
   ADVIES (klant, expert, gebied)
met de volgende functionele afhankelijkheden:
   1. klant, gebied → expert
   2. klant, expert → gebied
   3. expert → gebied

Mogelijke décompositities van deze relatie zijn:
1. ADVIES (klant, gebied)
   SPECIALIST (expert, gebied)
   waarbij verlies van informatie optreedt, n.l. van welke expert de klant advies krijgt.
   Dit is in overeenstemming met de stelling van Rissanen, daar het gemeenschappelijk attribuut in geen van beide relaties een kandidaatsleutel is.
2. ADVIES (klant, gebied)
   ADVISEUR (klant, expert)
   waarbij verlies van informatie optreedt, n.l. op welk gebied een expert kan adviseren.
   Ook dit is in overeenstemming met de stelling van Rissanen daar geen van de drie bovengenoemde functionele afhankelijkheden logisch afleidbaar is.
3. ADVISEUR (klant, expert)
   SPECIALIST (expert, gebied)
   waarbij geen verlies van informatie optreedt.
   Uit de functionele afhankelijkheid expert → gebied is de functionele afhankelijkheid klant, expert → gebied afleidbaar. Echter de functionele

afhankelijkheid klant, gebied → expert is niet afleidbaar m.a.w. volgens de stelling van Rissanen kunnen deze relaties niet onafhankelijk van elkaar worden gewijzigd. Een voorbeeld van een dergelijke wijziging is het opvoeren van een tupel van ADVISEUR (zie 5.7.5).

5.9. Vraagstukken

1. Touroperator

Een touroperator maakt voor het verzorgen van busreizen naar verschillende wintersportgebieden in Europa gebruik van een database. De touroperator beschikt over een groot aantal bussen die van elkaar worden onderscheiden door een bus#. Voor iedere bus wordt een rittenschema gemaakt. Een rit voert van of naar de hotels in een gebied. Een bus maakt op één dag hoogstens één rit van of naar een gebied. Het is niet ongebruikelijk dat in drukke perioden op hetzelfde gebied al dan niet op dezelfde dag meer dan één bus wordt ingezet. Het aantal personen dat met een bus kan worden vervoerd is niet voor alle bussen hetzelfde.

In ieder hotel reserveert de touroperator een aantal kamers van verschillende soorten (b.v. 1 pers. met bad, 2 pers., 3 pers. met douche). Bij de keuze van een hotel speelt de hoogte waarop het is gelegen en de klasse een belangrijke rol.

Iemand boekt voor zichzelf en mogelijk ook voor een of meer anderen een reis naar een bepaald hotel. Deze boeking wordt voorzien van een boeking#. In het hotel zijn voor deze boeking kamers van verschillende soorten nodig. Tevens worden de datum van de heenreis en de datum van de terugreis vastgesteld.
De kosten van een reis zijn samengesteld uit de kosten van het vervoer en de kosten voor het logies. De vervoerskosten worden berekend op basis van de prijs voor het vervoer van en naar het gebied per persoon. De logieskosten worden berekend op basis van de prijs van de betreffende kamer(s).
Deze prijzen veranderen gedurende het seizoen niet.

Ontwerp een relatiemodel in BCNF voor deze database. Gebruik alleen attributen die in de omschrijving worden genoemd. Geef van elke relatie de kandidaatsleutel(s).

2. Productie

Een bedrijf produceert artikelen. De artikelen worden van elkaar onderscheiden door een art# en hebben een artikelnaam. Steeds wordt getracht de voorraad van een artikel hoger te laten zijn dan de gewenste minimumvoorraad.
Het bedrijf heeft klanten die artikelen bestellen. De klanten worden van elkaar onderscheiden door een klantnummer. Klanten hebben een naam, adres, woonplaats, een kredietlimiet, een debetbedrag en één of meer afleveradressen (≡ straatnaam, nummer, plaatsnaam). Deze afleveradressen worden over de klanten heen van elkaar onderscheiden door het afadr#. Goederen kunnen op een afleveradres in ontvangst worden genomen door een of meer personen waarvan de namen (uniek per afleveradres) bekend zijn bij het bedrijf. Op een afleveradres kunnen geen goederen van verschillende klanten worden bezorgd.
Een klant plaatst op één dag hoogstens één bestelling. Een bestelling heeft een b# (uniek), een datum, een klant, een afleveradres en de hoeveelheid en prijs van de bestelde artikelen.
Indien de voorraad niet voldoende is om de bestelde hoeveelheid in één keer af te leveren vindt partiële aflevering plaats. Bijgehouden wordt dan hoeveel er nog moet worden geleverd.

Ontwerp een relatiemodel in BCNF voor deze database. Geef van elke relatie de kandidaatsleutel(s).

3. Organisatiestructuur.

Een bedrijf legt in een database gegevens vast over de structuur van de organisatie, de werknemers en de projecten die worden uitgevoerd.

De afdelingen zijn verdeeld in kleinere afdelingen die op hun beurt ook weer onderverdeeld kunnen zijn. Iedere afdeling heeft een budget. Van een medewerker wordt naam, adres en woonplaats vastgelegd. Iedere medewerker behoort tot één afdeling. De chef van de afdeling is tevens de hierarchische chef van de werknemer. Het bedrijf heeft projecten onder handen waaraan medewerkers voor een gedeelte van hun tijd toegevoegd zijn. Van een project wordt vastgelegd de naam, de opdrachtgever en de einddatum. Een van de medewerkers is de projectleider van het project. Vastgelegd wordt van welke projecten een medewerker projectleider is. Niet iedere medewerker van een afdeling kan worden ingezet voor projectwerk. Van de medewerkers die wel kunnen worden

ingezet wordt vastgelegd het max. aantal uren dat ze aan projectwerk mogen besteden en per project de tijd die reeds is besteed. Iedere medewerker heeft een functie.

Ontwerp een relatiemodel in BCNF voor deze database. Geef van elke relatie de kandidaatsleutel(s).

4. Bevolking

Het ministerie van Binnenlandse Zaken ontwerpt een database waarin de gegevens over plaatsen, personen en scholen worden opgeslagen.
Van iedere plaats is het gewenste aantal inwoners vastgesteld. Een plaatsnaam is uniek binnen een provincie. Iedere persoon krijgt een pers# (uniek). Van iedere persoon wordt opgenomen naam, adres, woonplaats, geboortedatum en geslacht. Personen voeren een gemeenschappelijke huishouding. Een van deze personen is het hoofd van het huishouden. Iedere persoon behoort tot één huishouden.
Sommige personen gaan naar school. Een persoon kan niet op twee of meer scholen tegelijk zitten.

Scholen worden van elkaar onderscheiden door de naam en de plaats waarin de school is gevestigd. Een school heeft een vastgesteld budget.

Ontwerp een relatiemodel in BCNF voor deze database. Geef van elke relatie de kandidaatsleutel(s).

5. Bibliotheek

Een bibliotheek ontwerpt een database ten behoeve van de uitleenadministratie en de systeemcatalogus.
De systeemcatalogus is een opsomming van onderwerpen. Ieder onderwerp heeft een systeemnummer, een naam en een omschrijving. Een titel is door één schrijver geschreven en wordt door één uitgever gedrukt en uitgegeven. De filialen (uitleencentra) kunnen van een titel één of meer boeken aanschaffen. Zo'n titel wordt dan in de systeemcatalogus opgenomen onder tenminste één onderwerp. Per filiaal krijgt ieder aangeschaft boek een uniek boeknummer. Voor de aanschaf beschikken de filialen over een eigen budget. Bijgehouden wordt hoeveel geld tot nu toe besteed is.

Lezers krijgen een lezerspas waarop lezersnummer, naam, adres en soort lidmaatschap staan vermeld. Van de uitgeleende boeken worden lezer en datum genoteerd. Indien een lezer een boek niet terugbrengt krijgt deze een boete ter grootte van de aanschafprijs.

Ontwerp een relatiemodel in BCNF voor deze database. Geef van elke relatie de kandidaatsleutel(s).

6. Bouwonderneming

Een bouwonderneming ontwerpt een database waarin gegevens over projecten, materialen en leveranciers worden opgenomen. Projecten worden van elkaar onderscheiden door hun naam; materialen en leveranciers door een nummer. Van materialen wordt tevens de naam en van leveranciers de NAW-gegevens opgenomen in de database. Opdrachtgevers van projecten vragen de bouwonderneming op basis van een bestek een offerte uit te brengen. In het bestek van het project is vastgelegd hoeveel van ieder materiaal nodig is. Bij het opstellen van een offerte wordt voor elk materiaal van een calculatieprijs (≡ de huidige gemiddelde prijs) per eenheid uitgegaan. Er wordt uitsluitend met materiaalkosten rekening gehouden. Tevens wordt vastgesteld in hoeveel tijd het project kan worden gerealiseerd.

Van de projecten die worden gerealiseerd worden de gegevens die nodig zijn om na te kunnen gaan of het project winstgevend is geweest, eveneens vastgelegd. De materialen worden geleverd door de leveranciers. Van een leverantie wordt ondermeer vastgelegd de datum, de hoeveelheden geleverde materialen en de prijs. Niet met alle leveranciers heeft de bouwonderneming even goede ervaringen. De kwaliteit van de leveranciers wordt daarom bijgehouden.

Ontwerp een relatiemodel in BCNF voor deze database. Geef van elke relatie de kandidaatsleutel(s).

7. Vliegreizen

Een luchtvaartmaatschappij maakt voor de registratie van passagiers voor haar vluchten gebruik van een database. Elke week wordt hetzelfde vluchtschema - dienstregeling van de vluchten - uitgevoerd.
Een vlucht heeft een vluchtnummer, een vertrek- en aankomsthaven, een ver-

trekdag (b.v. maandag) en een vertrektijdstip. Bij sommige vluchten worden in vaste volgorde tussenstops gemaakt. Omdat een vlucht altijd met hetzelfde type toestel wordt uitgevoerd, is het aantal beschikbare plaatsen op een vlucht steeds hetzelfde. Boekingen van passagiers lopen via reisbureaus. Reisbureaus, die van elkaar worden onderscheiden door een RB#, kunnen vooraf per vlucht voor één of meer data een aantal plaatsen reserveren. De luchtvaartmaatschappij beschikt over de telefoonnummers en adressen van de reisbureaus.
Iemand boekt voor zichzelf en mogelijk ook voor één of meer anderen voor een (gedeelte van een) vlucht. Het reisbureau gaat bij de luchtvaartmaatschappij na of er op de gewenste vertrekdatum voldoende plaatsen zijn. Is dit het geval dan wordt de boeking door de luchtvaartmaatschappij geaccepteerd, gedateerd en voorzien van een boekingsnummer. Bovendien worden om een passagierslijst te kunnen samenstellen de namen van alle geboekte passagiers geregistreerd.

Ontwerp een relatiemodel in BCNF voor deze database.
Geef van elke relatie de kandidaatsleutel(s).

8. **Autoverhuur**

Bij autoverhuurbedrijf Intercar kan men auto's huren uit verschillende prijsklassen. Afhankelijk van de prijsklasse betaalt men een vast bedrag per dag en een bepaalde kilometerprijs.

Als een toekomstige huurder zeker wil zijn vanaf een bepaalde datum gedurende een aantal dagen over een auto van een bepaalde prijsklasse te kunnen beschikken, kan hij reserveren. Reserveringen worden gedateerd. Indien later - bij de verhuur - door onvoorziene omstandigheden er toch geen auto in de betreffende klasse beschikbaar is, krijgt de huurder 10% korting op de prijs van de dan gehuurde auto.

Bij de verhuur wordt - indien dit al niet bij de reservering is gedaan - naam, adres en woonplaats en rijbewijs van de huurder genoteerd. De huurder kan vanaf deze dag een (verwacht) aantal dagen over een bepaalde auto beschikken.

Als een auto wordt teruggebracht wordt het aantal gereden kilometers gere-

gistreerd en het aantal dagen dat de auto is gebruikt. Bovendien wordt dan de kilometerstand in het systeem weer bijgewerkt.

Het bedrijf heeft zelf de auto's in onderhoud. Het moet op ieder moment mogelijk zijn om na te gaan wanneer een auto in onderhoud is geweest, bij welke kilometerstand, of een grote of kleine onderhoudsbeurt is gegeven of dat het een reparatie betrof, wat de kosten waren en hoeveel dagen er mee gemoeid waren.

Ontwerp een relatiemodel in BCNF voor een database voor bovenstaande toepassing. Geef van elke relatie de kandidaatsleutel(s).

9. Opleidingsinstituut

Een opleidingsinstituut gebruikt voor de planning van kursussen een database.
De kursussen hebben altijd betrekking op één van de modulen uit het opleidingsprogramma. De modulen worden van elkaar onderscheiden door een mod# en zijn voorzien van een naam. De kursussen worden van elkaar onderscheiden door een kursus#.
De kursussen worden gegeven door externe docenten. Het instituut heeft voor intern gebruik de docenten een nummer gegeven. Van een docent is verder bekend naam-, adresgegevens en voor welke modulen hij inzetbaar is. Het opleidingsinstituut houdt om een zo goed mogelijke opleiding te verzorgen de geschiktheid van een docent voor het geven van een module bij. Zodra de plaats (≡ plaatsnaam) waar de kursus gegeven wordt en de aanvangsdatum bekend zijn, kunnen kursisten zich voor de kursus aanmelden.

Een kursus gaat niet door indien het aantal aanmeldingen kleiner is dan het voor de module geldende minimum. Het aantal deelnemers voor een kursus is gelimiteerd. Naam en adres van de deelnemers worden opgenomen waarbij het niet in de bedoeling ligt van het opleidingsinstituut om een kursistenadministratie te voeren. Verder wordt voor iedere kursus een docent gezocht.

Ontwerp voor deze database een relatiemodel in BCNF. Geef van elke relatie de kandidaatsleutel(s).

10. Projectuitvoering

Een bedrijf voert projecten uit. Ieder project omvat een aantal aktiviteiten zoals analyse, ontwerp, bouw, enz. De toestand waarin een aktiviteit verkeert wordt aangegeven met 'P' (gepland), 'S' (gestart) of 'E' (beëindigd). Werknemers kunnen worden ingezet voor één of meer aktiviteiten. Van een inzetbare werknemer wordt het aantal uren dat hij kan worden ingezet voor een aktiviteit in de database opgenomen (dus uren-inzetbaar > 0). Voordat een aktiviteit wordt gestart, wordt een leider aangewezen.

Voor deze project database zijn de volgende relaties in BCNF ontworpen:

werknemer W(pers#, pers-naam, funktie, afdeling)
project P(proj#, proj-naam, proj-omschrijving)
aktiviteit A(proj#, akt#, pers#leider, toestand, begindatum)
inzet I(pers#, proj#, akt#, uren-inzetbaar, uren-besteed)

N.B. De attributen pers-naam en proj-naam zijn kandidaatsleutels.

1. Veronderstel dat relatie I niet in BCNF is maar in 2NF.
   a. Noem de functionele afhankelijkheden die dan moeten gelden.
   b. Noem de functionele afhankelijkheden die dan beslist niet gelden.

   Ga van onderstaande beweringen na of zij juist of onjuist zijn.
   Geef een kort doch goed gemotiveerd antwoord.

2. Een werknemer kan alleen maar voor de aktiviteiten van één project worden ingezet.
3. Een werknemer kan van meer dan één aktiviteit van hetzelfde project leider zijn.
4. De datum vanaf wanneer een aktiviteit van een project in een bepaalde toestand verkeert is bekend.

11. Makelaars

De bond van makelaars besluit ten behoeve van de koop en verkoop van huizen een database te gaan gebruiken.

Van makelaars wordt de kantoornaam (uniek), het kantooradres, de naam van de makelaar en het telefoonnummer opgenomen. Makelaars ontvangen van eigenaars van huizen gedateerde opdrachten tot verkoop. Een eigenaar kan één of meer andere makelaars opdracht tot verkoop geven indien de verkoop van het huis te lang gaat duren. Ook kan dan de vraagprijs worden verlaagd.

Kopers kunnen een bod uitbrengen op een huis bij één van de verkopende makelaars. Zij kunnen dit bod zelf uitbrengen of dat laten doen via de door hen ingeschakelde makelaar. Ieder bod wordt in de database opgenomen. De verdeling van de courtage gebeurt aan de hand van de kosten die de makelaars hebben gemaakt.

Voor deze database zijn de volgende relaties in BCNF ontworpen:

makelaar(kantoornaam, kantooradres, naam, telefoon)
huis(huisadres, eigenaar, vraagprijs)
verkoopopdracht(kantoornaam, huisadres, datum, kosten)
koopopdracht(koper, kantoornaam, datum)
bod(kantoornaam, huisadres, koper, datum, bedrag)

Ga van onderstaande beweringen na of zij juist of onjuist zijn. Geef een kort doch goed gemotiveerd antwoord.

1. Een koper kan de door hem ingeschakelde makelaar onder slechts één telefoonnummer bereiken.
2. Een koper kan niet twee of meer keren een bod doen op hetzelfde huis bij dezelfde makelaar.
3. Iedere keer als een makelaar kosten maakt voor een huis kunnen deze kosten in de database worden verwerkt.
4. Een makelaar die door een koper is ingeschakeld kan niet door een andere koper worden ingeschakeld.
5. Het opnemen van huisadres in de primary key zal bij het gebruik van de database problemen geven b.v. bij het opvoeren en raadplegen van tuples van bod.
6. De vraagprijs van een huis is voor alle makelaars waarbij het in de verkoop staat gelijk.
7. De kosten die een makelaar maakt voor een huis worden per dag geregistreerd.

8. Wanneer een koper een nieuw bod uitbrengt op een huis dan wordt dit bod bij dezelfde verkopende makelaar uitgebracht.
9. Wanneer een koper een nieuw bod laat uitbrengen op een huis dan wordt dit bod door dezelfde kopende makelaar uitgebracht.

5.10. Nabeschouwing

Wanneer we het relationele datamodel vergelijken met andere modellen, dan kunnen we vaststellen dat dit model wat de wiskundige fundering betreft gunstig afsteekt. Zoals al eerder is opgemerkt is dit een gevolg van het feit dat het relationele model niet is afgeleid uit de praktijk van de gegevensverwerking. Eerst is de theorie ontwikkeld en daarna zijn er implementaties gekomen.

We mogen stellen dat de komst van het relationele model een geweldige stimulans is geweest voor het denken over gegevens en hun logische samenhang, getuige ook de stroom van artikelen die op dit terrein verschenen is.

Dat men, ook in het geval dat men niet de beschikking heeft over een relationeel database management systeem, de theorie van het relationele model kan gebruiken bij het opstellen van een conceptueel model, wordt wel als positief punt van het relationele model gezien. Uiteraard moet in zo'n situatie nog een "vertaalstap" worden gemaakt naar het conceptuele model behorende bij het ter beschikking staande systeem. Het voordeel is echter dat het eerstgenoemde conceptuele model systematisch kan worden ontworpen.

Toch is ook het relationele model niet vrij van problemen. Zonder er diep op in te gaan willen we er enige noemen. Zoals we hebben gezien kan het normaliseren tot het uiterste in de praktijk soms leiden tot minder bevredigende relaties. Dit probleem heeft vooral te maken met het mogen voorkomen van samengestelde sleutels (eng.: compound keys).

Een voorbeeld hiervan is het probleem dat bekend staat onder de naam connection trap. Indien twee relaties een attribuut gemeenschappelijk hebben, kan door samenvoeging (join, zie relationele algebra) een relatie worden gecreëerd waaraan een betekenis kan worden gegeven die deze niet heeft.

Als laatste noemen we het ontbreken van de mogelijkheid tot specialisatie. Veronderstel dat in een bedrijf gegevens over werknemers moeten worden vastgelegd, waarbij twee categorieën werknemers zijn te onderscheiden. Van de werknemers die op kantoor werken worden andere gegevens opgenomen dan van die uit de fabriek. Gedeeltelijk zijn de gegevens hetzelfde (b.v. naw, geb.dat., etc). Het relationele model biedt dan de volgende mogelijkheden:

- één relatie WERKNEMER met alle voorkomende attributen, óf
- twee relaties WKANT en WFAB met de passende attributen.

Ga zelf de consequenties van deze oplossingen na.

De geschetste en andere tekortkomingen van het relationele model hebben geleid tot het semantische model (zie 4.1.), waarop in een afzonderlijk college verder wordt ingegaan.

6. RELATIONELE ALGEBRA

6.1. Inleiding

De Relationele Algebra is een verzameling operatoren op relaties. Elke operator heeft één of twee relaties als operand en levert als resultaat weer een relatie op. Deze relatie kan dus weer dienen als operand. Tot de operatoren behoren de selectie, projectie, join, deling en de bekende verzamelingsoperatoren vereniging, doorsnede, verschil en cartesisch produkt.
De relationele algebra kan de bouwstenen leveren voor een hogere taal waarin het opvragen van gegevens uit een relationele database exact en bondig kan worden geformuleerd.

6.2. Operaties

In deze paragraaf worden voorbeelden gegeven van operaties die betrekking hebben op de volgende relationele database.

Een bedrijf bestaat uit fabrieken die artikelen produceren. Steeds wordt er naar gestreefd dat de voorraad van een artikel in een fabriek groter is dan de gewenste minimum voorraad. Klanten bestellen op een bestelling bij een fabriek één of meer artikelen in een bepaalde hoeveelheid en tegen een zekere prijs. De datum van de bestelling wordt in de database opgenomen. Het relationele schema van deze database is:

F(fab#, art#, vrd, minvrd)
B(b#, klant, fab#, datum)
BA(b#, art#, hoev, prijs)

De database bevat de volgende tupels:

F

fab #	art #	vrd	minvrd
F1	A1	200	150
F1	A2	300	75
F1	A4	150	300
F2	A1	700	200
F2	A2	875	100
F2	A3	0	20
F2	A4	210	210
F2	A5	370	180
F3	A3	15	30
F3	A4	175	100
F4	A4	300	100

B

b #	klant	fab #	datum
B1	Smit	F4	16-04-83
B2	Slager	F3	25-08-83
B3	Snoek	F2	17-10-83
B4	Staal	F1	04-05-83
B5	Slager	F2	19-11-83
B6	Smit	F2	17-10-83

BA

b #	art #	hoev	prijs
B1	A4	200	4.50
B2	A3	150	3.50
B2	A4	75	4.75
B3	A1	200	2.75
B3	A2	100	2.--
B3	A3	100	3.50
B3	A4	50	5.--
B3	A5	1000	0.50
B4	A2	300	2.--
B4	A4	100	5.--
B5	A1	200	2.75
B5	A2	300	1.75
B5	A5	400	0.50
B6	A1	400	2.50
B6	A3	200	3.25

6.2.1. Selectie

Functie:
Selecteren van de tupels van een relatie waarvan de waarde van een attribuut aan een bepaalde voorwaarde voldoet. De voorwaarde betreft het vergelijken met een constante. De constante moet tot het domein van het attribuut horen.

Gegeven de relatie R(A,B,C). Een willekeurige tupel uit R duiden we aan met r. De waarde van het attribuut A van tupel r met $r[A]$.
Definitie Selectie

$$R[A\Theta\alpha] = \{r : r \in R \wedge r[A]\Theta\alpha\}$$

waarin α een constante is en Θ één van de relatieoperatoren $=$, $\neq$, $<$, $\leq$, $>$ of $\geq$

Voorbeelden

1. De voorraadgegevens van fabriek F3 worden verkregen met de opdracht:

   $F[fab\# = F3]$

   Resultaat:

fab #	art #	vrd	minvrd
F3	A3	15	30
F3	A4	175	100

2. De gegevens van de bestellingen die geplaatst zijn voor 1 augustus 83 worden verkregen met:

   $B[datum < 1\text{-}8\text{-}83]$

   Resultaat:

b #	klant	fab #	datum
B1	Smit	F4	16-04-83
B4	Staal	F1	04-05-83

6.2.2. Restrictie

Functie:
Selecteren van de tupels van een relatie waarvan de waarden van twee verschillende attributen aan een bepaalde voorwaarde voldoen. De voorwaarde betreft het onderling vergelijken van de attribuutwaarden. Deze waarden kunnen alleen worden vergeleken indien zij op hetzelfde domein zijn gedefinieerd.

Gegeven de relatie R(A,B,C) waarbij B en C op hetzelfde domein zijn gedefinieerd.
Definitie Restrictie:

$$R[B\Theta C] = \{r : r \in R \wedge r[B] \Theta r[C]\}$$

waarin Θ één van de relatieoperatoren $=$, $\neq$, $<$, $\leqslant$, $>$ of $\geqslant$ is.

Voorbeeld.

1. De voorraadgegevens van de fabrieken waarbij de voorraad van een artikel kleiner is dan de gewenste voorraad.

   $F[vrd < minvrd]$

   Resultaat:

fab #	art #	vrd	minvrd
F1	A4	150	300
F2	A3	0	20
F3	A3	15	30

6.2.3. Projectie

Functie.
Selecteren van de waarden van de gespecificeerde attributen van alle tupels van een relatie, met weglating van eventueel ontstane duplicaten.

Gegeven de relatie R(A,B,C)
Definitie Projectie:

$$R[A] = \{r[A] : r \in R\}$$

Voorbeelden

1. De klanten die tenminste één bestelling hebben gedaan:

   B[klant]

   Resultaat:

klant
Smit
Slager
Snoek
Staal

2. De artikelen en bijbehorende prijs die op bestelling B5 zijn besteld:

   BA[b# = B5] [art#, prijs]

   Resultaat

art #	prijs
A1	2.75
A2	1.75
A5	0.50

We merken op dat de operaties van links naar rechts worden afgehandeld. In dit geval wordt dus eerst de selectie uitgevoerd en daarna op het met de selectie verkregen resultaat, een projectie.

6.2.4. Join

Functie:

Selecteren van combinaties van de tupels van twee relaties. Een combinatie ontstaat door uit beide relaties een tupel te nemen en deze achter elkaar te plaatsen. Tot het resultaat behoren slechts die combinaties waarvan de waarden van twee attributen, één uit elke relatie, aan een bepaalde voorwaarde voldoen. De voorwaarde betreft het onderling vergelijken van deze attribuutwaarden.

Gegeven de relaties R(A,B,C) en S(D,E) waarbij C en D op hetzelfde domein zijn gedefinieerd. Een combinatie van een tupel r met een tupel s duiden we aan met (r,s).

Definitie Join:

$R[C\Theta D]S = \{(r,s) : r \in R \wedge s \in S \wedge r[C]\Theta s[D]\}$

waarin Θ één van de relatieoperatoren is.

Voorbeelden.

1. De klanten die artikel A2 hebben besteld.

   Deze klanten kunnen b.v. worden verkregen door een selectie van BA op art# = A2, daarna een join met B op b#, gevolgd door een projectie op klant, dus:

   $BA[art\# = A2]\ [b\# = b\#]B[klant]$

   Resultaat:

klant
Snoek
Staal
Slager

Bij toename van het aantal operaties in een opdracht wordt al snel het overzicht verloren, waardoor de kans op het maken van fouten groter wordt. Bedoeld worden hier "semantische fouten", waarop in tegenstelling tot syntactische fouten geen foutmelding volgt. Men krijgt dan i.h.a. wel een resultaat - dat er soms zeer aannemelijk uitziet - maar wel bij een andere vraag hoort dan die men bedoeld had te stellen. Dit kan worden vermeden door gebruik te maken van hulprelaties. Het proces van het tot stand komen van het resultaat kan daarmee doorzichtig worden gemaakt.

De klanten die artikel A2 hebben besteld kunnen ook worden verkregen met de opdrachten:

$R := BA[art\# = A2]$

$R[b\# = b\#]\ B\ [klant]$

waarin de hulprelatie R de b#'s van bestellingen van artikel A2 bevat.

2. De levering van bestelling B2 stagneert. Er moet daarom een overzicht worden gemaakt van de bestelde artikelen en de fabrieken die ze produceren:

   R := BA[b# = B2]

   R[art# = art#] F [art#, fab#]

   Resultaat:

art #	fab #
A3	F2
A3	F3
A4	F1
A4	F2
A4	F3
A4	F4

6.2.5. Vereniging

Functie:
Selecteren van de tupels van twee relaties die in de ene of in de andere relatie voorkomen. De relaties moeten daarbij vergelijkbaar zijn m.b.t. alle attributen. Bij elk attribuut in de ene relatie hoort een attribuut in de andere relatie dat op hetzelfde domein is gedefinieerd.

Gegeven de relaties R(A,B,C) en S(D,E,F) waarbij A en D op hetzelfde domein, B en E op hetzelfde domein en C en F op hetzelfde domein zijn gedefinieerd.

Definitie Vereniging:

$R \cup S = \{r : r \in R \vee r \in S\}$

Voorbeeld.

1. De voorraadgegevens van fabrieken die artikel A3 of A5 produceren.

   F[art# = A3] ∪ F[art# = A5]

Resultaat:

fab #	art #	vrd	minvrd
F2	A3	0	20
F3	A3	15	30
F2	A5	370	180

6.2.6. Doorsnede

Functie:
Selecteren van de tupels van twee relaties die in de ene én in de andere relatie voorkomen. De relaties moeten daarbij vergelijkbaar zijn.

Gegeven de relaties R(A,B,C) en S(D,E,F) die vergelijkbaar zijn.

Definitie Doorsnede:

$R \cap S = \{r : r \in R \wedge r \in S\}$

Voorbeeld.

1. De fabrieken die artikel A3 en A5 produceren.

   $F[art\#=A3][fab\#] \cap F[art\#=A5][fab\#]$

   Resultaat:

fab #
F2

6.2.7. Verschil

Functie:
Selecteren van de tupels van een relatie die wel in deze relatie maar niet in een andere relatie voorkomen. De relaties moeten daarbij vergelijkbaar zijn.

Gegeven de relaties R(A,B,C) en S(D,E,F) die vergelijkbaar zijn.

Definitie Verschil:

$R-S = \{r : r \in R \wedge r \notin S\}$

Voorbeeld.

1. De fabrieken die wel artikel A3 maar niet artikel A5 produceren.

   F[art#=A3][fab#] - F[art#=A5][fab#]

   Resultaat:

fab #
F3

6.2.8. Deling

Functie:
Selecteren van de tupels van een relatie die voldoen aan de voorwaarde dat de verzameling van waarden van een gespecificeerd attribuut in de tupels met gelijke waarde voor de niet gespecificeerde attributen, de verzameling van waarden van een ander attribuut omvat. De attributen moeten op hetzelfde domein zijn gedefinieerd.

Gegeven de relaties R(A,B,C) en S(D,E) waarbij C en D op hetzelfde domein zijn gedefinieerd.

Definitie Deling:

$R[C \div D]S = \{r[A,B] : r \in R \wedge \forall_{s \in S} (r[A,B], s[D]) \in R\}$

d.w.z.: de deling $R[C \div D]S$ levert die combinaties van A en B op ($r[A,B]$), waarvoor geldt dat r voorkomt in R ($r \in R$) en dat bovendien voor alle tupels s ($s \in S$) geldt dat de combinatie ($r[A,B]$, $s[D]$) voorkomt in R.

Voorbeelden.

1. De bestellingen waarop alle te produceren artikelen zijn besteld.

Een bestelling voldoet indien de verzameling van de op die bestelling bestelde artikelen de verzameling van alle te produceren artikelen omvat. Alle combinaties van b# en art# worden verkregen door een projectie van BA op b#,art#. Alle te produceren artikelen vinden we in de relatie F. De b# van bestellingen waarbij al deze artikelen zijn besteld worden dan verkregen door een deling.

R := BA[b#, art#]

R[art# ÷ art#] F

{De verzamelingen van waarden van art#'s in BA bij gelijke waarde van b# worden vergeleken met de verzameling van waarden van art#'s in F}

Resultaat:

b #
B3

Opgemerkt wordt dat de gevraagde bestellingen niet worden verkregen met

BA[art# ÷ art#] F

Hiermee worden nl. de bestellingen geselecteerd waarop alle artikelen in dezelfde hoeveelheid en tegen dezelfde prijs zijn besteld.

Een projectie van de relatie waarop wordt gedeeld is bijna altijd noodzakelijk. Deze projectie bevat tenminste de gevraagde attributen en het attribuut waarover wordt gedeeld.

Een projectie van de relatie waardoor wordt gedeeld is niet noodzakelijk, omdat volgens de definitie alleen het genoemde attribuut van de "deler" een rol speelt.

2. De artikelen die bij alle fabrieken worden geproduceerd.

Een artikel voldoet indien de verzameling van fabrieken die het artikel produceren de verzameling van fabrieken omvat.

Alle combinaties van fab# en art# worden verkregen door een projectie van F op fab#,art#. Alle fabrieken vinden we in de relatie F. De gevraagde artikelen worden dan verkregen door een deling.

R := F[art# , fab#]

R[fab# ÷ fab#] F

Resultaat:

art #
A4

6.2.9. Alias

Functie:
Door alleen een attribuutnaam te noemen is het niet altijd duidelijk wat de bijbehorende relatie van het attribuut is. In gevallen dat hierover misverstanden kunnen ontstaan, wordt de relatienaam als prefix gebruikt:

relatienaam.attribuutnaam

Toch zijn er situaties waarin een attribuut ook op deze wijze niet eenduidig wordt aangewezen o.a. bij een join van een relatie met zichzelf. In dit soort situaties is het noodzakelijk om relaties te herbenoemen.

Voorbeeld.

1. De combinaties van klanten die op dezelfde dag een bestelling hebben geplaatst.

   R ALIAS FOR B

   B[datum = datum] R[B.klant, R.klant]

   Resultaat:

B.klant	R.klant
Snoek	Snoek
Snoek	Smit
Smit	Snoek
Smit	Smit
Slager	Slager
Staal	Staal

Dit resultaat bevat een aantal volstrekt overbodige tupels.
Op de eerste plaats zijn dit de tupels die geen informatie geven omdat de-

zelfde klant twee keer wordt genoemd b.v. Smit, Smit. Op de tweede plaats zijn dit de tupels die dezelfde informatie nogmaals geven omdat dezelfde twee klanten worden genoemd b.v. Smit, Snoek.
De overbodige tupels worden verwijderd door op het resultaat een restrictie toe te passen:

```
R ALIAS FOR B
   S:=B[ datum=datum] R [B.klant, R.klant]
   S[B.klant < R.klant]
   Resultaat:
```

B.klant	R.klant
Smit	Snoek

6.3. Vraagstukken

1. **Fabrieken**

Geef opdrachten in Relationele Algebra voor het verkrijgen van de volgende gegevens uit de in 6.2 beschreven database:

F(fab#,art#,vrd,minvrd)
B(b#,klant,fab#,datum)
BA(b#,art#,hoev,prijs)

1. De fabrieken die artikel A5 niet produceren.

2. De fabrieken die tenminste een ander artikel dan artikel A5 produceren.

3. De fabrieken die alleen artikel A5 produceren.

4. De fabrieken die artikel A5 en tenminste een ander artikel produceren.

5. De klanten, de artikelen en bijbehorende hoeveelheden waarin de klant het artikel bij fabriek F3 heeft besteld.

6. De klanten die artikel A2 voor minder dan f. 2,-- hebben besteld.

7. De klanten die alleen artikel A4 hebben besteld.

8. De klanten die tenminste twee bestellingen hebben geplaatst.

9. De bestellingen waarop slechts één artikel is besteld.

10. De artikelen, de klant en de fabriek waar de voorraad van het bestelde artikel kleiner is dan de bestelde hoeveelheid.

11. De klanten die artikel A1 voor een lagere prijs hebben besteld dan waarvoor A1 op bestelling B3 is besteld.

12. De artikelen die door alle klanten zijn besteld.

13. De klanten die op één bestelling tenminste de artikelen van bestelling B1 hebben besteld.

14. De overige fabrieken die tenminste die artikelen kunnen produceren, die door fabriek F1 worden geproduceerd.

15. De artikelen die op alle bestellingen voor dezelfde prijs zijn besteld.

16. De overige klanten die tenminste de door klant Staal bestelde artikelen hebben besteld.

2. **Leveranties**

De volgende vragen hebben betrekking op de volgende database:

leverancier: L(L#, Lnaam, Lwpl)
artikel: A(A#, Anaam, Asoort)
project: P(P#, Pnaam, Pplaats)
levering: LAP(L#, A#, P#, datum, hoev)

1. De namen van de leveranciers die aan een project in Oss cement hebben geleverd.

2. De namen en plaatsen van de projecten waaraan tenminste de door L1 geleverde artikelen zijn geleverd.

3. De namen van de leveranciers die eenzelfde artikel aan alle projecten in Putten hebben geleverd.

4. De namen van de projecten waaraan alle leveringen door één leverancier zijn verricht.

5. De namen van de leveranciers die aan project Hoogbouw tenminste twee verschillende artikelen hebben geleverd.

3. Makelaars

De bond van makelaars besluit ten behoeve van de koop en verkoop van huizen een database te gaan gebruiken. Van makelaars wordt de kantoornaam (uniek), het kantooradres, de naam van de makelaar en het telefoonnummer opgenomen. Makelaars ontvangen van eigenaars van huizen gedateerde opdrachten tot verkoop. Een eigenaar kan één of meer andere makelaars opdracht tot verkoop geven indien de verkoop van het huis te lang gaat duren. Ook kan dan de vraagprijs worden verlaagd.

Kopers kunnen een bod uitbrengen op een huis bij één van de verkopende makelaars. Zij kunnen dit bod zelf uitbrengen of dat laten doen via de door hen ingeschakelde makelaar. Ieder bod wordt in de database opgenomen. De verdeling van de courtage gebeurt aan de hand van de kosten die de makelaars hebben gemaakt.

Voor deze database zijn de volgende relaties in BCNF ontworpen:

makelaar(kantoornaam, kantooradres, naam, telefoon)
huis(huisadres, eigenaar, vraagprijs)
verkoopopdracht(kantoornaam, huisadres, datum, kosten)
koopopdracht(koper, kantoornaam, datum)
bod(kantoornaam, huisadres, koper, datum, bedrag)

Geef de opdrachten in Algebra voor het verkrijgen van de volgende gegevens:

1. De namen en telefoonnummers van de makelaars waaraan na 01.08.1983 huizen met een vraagprijs beneden de fl. 300.000,-- in de verkoop zijn gegeven en waarop bij hem tenminste één bod boven de fl. 250.000,-- is uitgebracht.

2. De namen en telefoonnummers van de makelaars waarbij door eenzelfde koper op alle huizen van eigenaar Huisboer een bod van tenminste fl. 250.000,-- is uitgebracht.

3. De eigenaars van de huizen waarop door precies één koper is geboden.

7. SQL

7.1. Inleiding

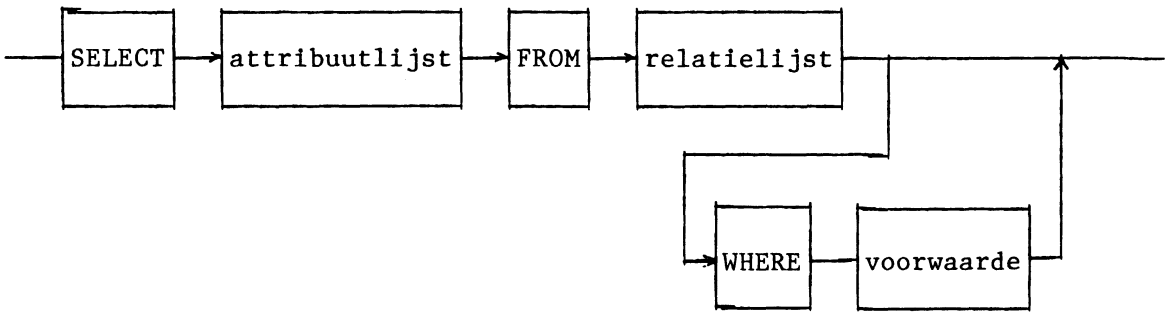
SQL (Structured Query Language) is een taal die gebaseerd is op het relationele model. Zij is oorspronkelijk ontworpen door IBM (D.D. Chamberlin e.a.). Momenteel bestaan er implementaties voor verschillende relationele database managementsystemen. SQL mag zich verheugen in een toenemende belangstelling.
Hoewel uit de naam zou kunnen worden afgeleid dat SQL alleen een vraagtaal is, kan zij ook voor database-creatie, gegevensmanipulatie en het definiëren van views (enigszins vergelijkbaar met externe schema's) worden gebruikt. In dit hoofdstuk wordt van de manipulatie-opdrachten SELECT, UPDATE, DELETE en INSERT alleen de SELECT-opdracht behandeld. Enige niet essentiële zaken zijn weggelaten.

7.2. Syntaxis van de SELECT opdracht

Met een SELECT opdracht worden alle tupels van een of meer relaties benaderd. Van de benaderde tupels worden diegenen die aan een voorwaarde voldoen geselecteerd. De waarden van de gespecificeerde attributen van de geselecteerde tupels vormen de tupels van het resultaat. Het resultaat is dus zelf weer een relatie en bevat derhalve geen gelijke tupels.

De syntaxis van de SELECT opdracht wordt weergegeven met het volgende diagram:

selectopdracht

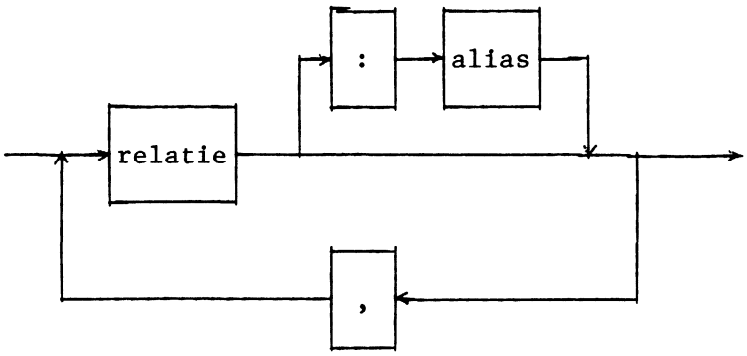


Dit soort diagrammen moet als volgt worden gelezen. Indien de tekst in een

rechthoek met hoofdletters is geschreven of een symbool is, dan moet de tekst letterlijk in de opdracht worden opgenomen. Tekst geschreven met kleine letters wordt gedefinieerd door ðf een diagram met dezelfde naam ðf m.b.v. een beschrijving. De lijnen moeten worden gevolgd in de richting van de pijlen. Bij een vertakking kan een keuze worden gemaakt.

De relatielijst wordt gedefinieerd door het diagram:

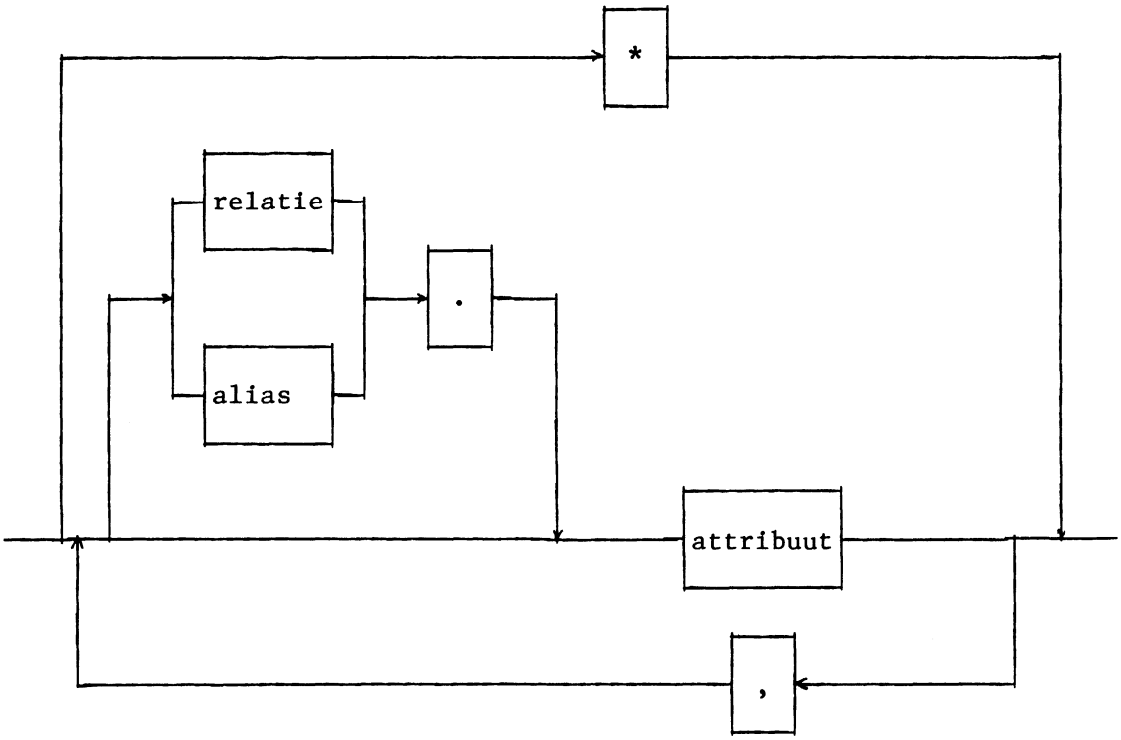
relatielijst



Hierin is relatie een van de relaties in de database. Alias is de naam waarmee de voorafgaande relatie in het vervolg van de selectopdracht ook aangeduid mag worden. De tupels van de relaties genoemd in de relatielijst worden één voor één gecombineerd benaderd.

De attribuutlijst wordt gedefinieerd door het diagram:

attribuutlijst

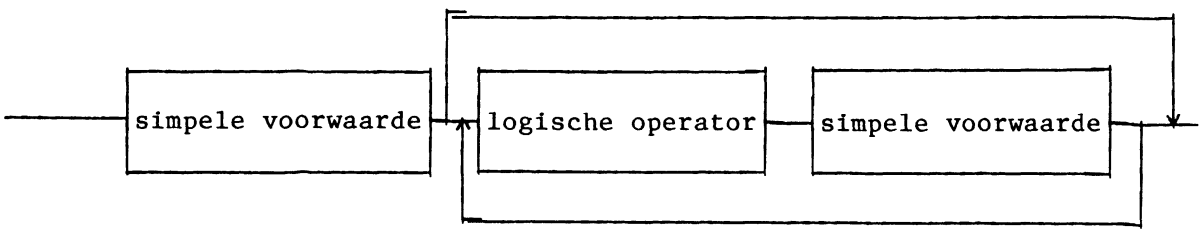


De attribuutwaarden van de attributen genoemd in deze lijst vormen de tupels van het resultaat. Ieder attribuut moet deel uitmaken van precies één relatie uit de relatielijst. Indien * wordt gekozen dan wordt het resultaat gevormd door alle attributen van de relaties uit de relatielijst.

De voorwaarde is een predikaat waaraan een benaderd tupel of combinatie al dan niet voldoet. De tupels die er aan voldoen vormen het resultaat van de select opdracht. Als geen voorwaarde is opgenomen dan vormen alle benaderde tupels het resultaat.

Een voorwaarde kan zijn opgebouwd uit simpele voorwaarden. Het syntaxisdiagram hiervan luidt:

voorwaarde



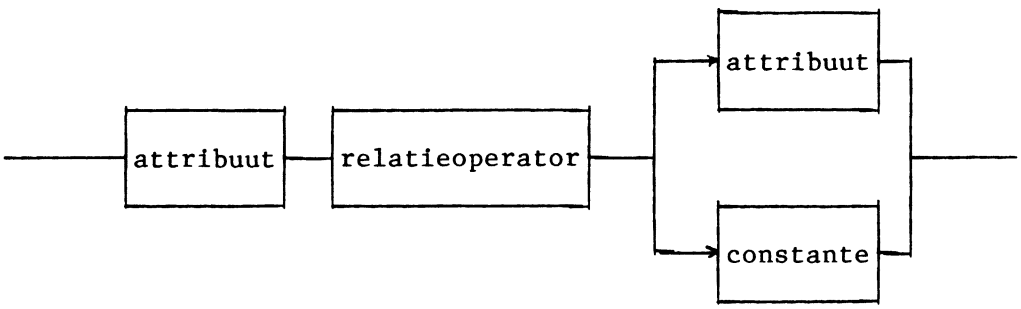
Hierin is logische operator een van de operatoren AND of OR en simpele voorwaarde een attribuutvoorwaarde, een tupelvoorwaarde of een relatievoorwaarde.

In een voorwaarde waar meer dan één simpele voorwaarde voorkomt, hebben de relatie operatoren een hogere prioriteit dan de logische operatoren. Operatoren van gelijke prioriteit worden van links naar rechts uitgevoerd. Door middel van ronde haken kan de volgorde van uitvoering op de gebruikelijke manier worden beïnvloed.

We onderscheiden drie soorten simpele voorwaarden n.l.:
1. attribuutvoorwaarde
2. tupelvoorwaarde
3. relatievoorwaarde.

Een attribuutvoorwaarde is een vergelijking van de waarde van een attribuut met een constante of met de waarde van een ander attribuut. Het syntaxisdiagram luidt:

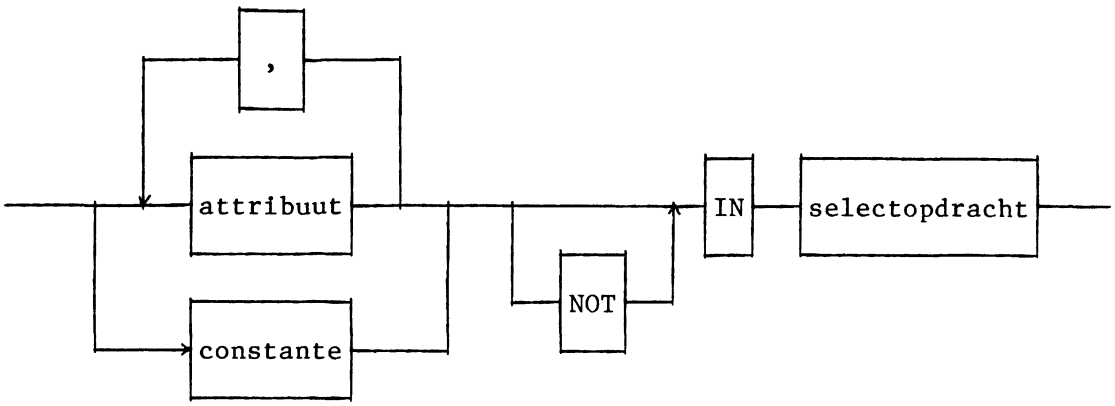
attribuutvoorwaarde



Hierin is de relatieoperator een van de operatoren =, ≠, <, ≤, >, of ≥. De attributen of het attribuut en de constante moeten op hetzelfde domein zijn gedefinieerd.

Een tupelvoorwaarde is de vraag of een (combinatie van) attribuutwaarde(n) of een constante een tupel is in een relatie.

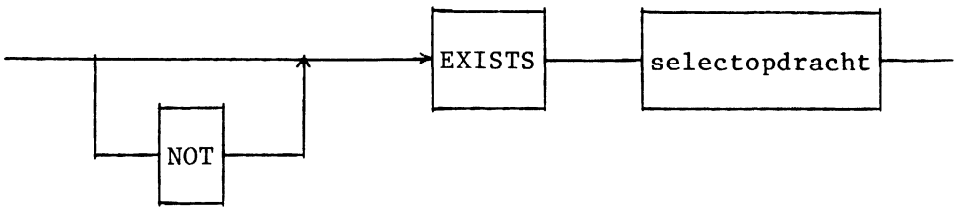
tupelvoorwaarde



Door de selectopdracht wordt een relatie bepaald. De attributen van deze relatie moeten op dezelfde domeinen zijn gedefinieerd als de genoemde attributen. Als een constante wordt gebruikt dan moet deze behoren tot hetzelfde domein als het attribuut van de relatie. Aan de voorwaarde wordt voldaan indien de (combinatie van) attribuutwaarde(n) of constante al of niet deel uitmaakt van de relatie, afhankelijk van het gebruik van NOT.

Een relatievoorwaarde is de vraag of een relatie al dan niet tupels bevat. Het syntaxisdiagram hiervan is:

relatievoorwaarde



Door de selectopdracht wordt een relatie bepaald. Aan de voorwaarde wordt voldaan indien deze relatie tenminste één tupel of geen tupels bevat, afhankelijk van het gebruik van NOT.

7.3. Voorbeelden

In deze paragraaf worden voorbeelden van de SELECT-opdracht gegeven die betrekking hebben op de relationele database gegeven in paragraaf 6.2.

Het relationele schema van deze database is:
F(fab#, art#, vrd, minvrd)
B(b#, klant, fab#, datum)
BA(b#, art#, hoev, prijs)

7.3.1. Selecteren zonder voorwaarde

Met een select zonder voorwaarde worden alle (combinaties van de) tupels van de relatie(s) geselecteerd. Het resultaat bevat de waarde van de gespecificeerde attributen van de geselecteerde tupels.

Voorbeelden

1. De klanten:
   SELECT klant
   FROM B
   Resultaat:

klant
Smit
Slager
Snoek
Staal

2. De voorraadgegevens van de fabrieken:
   SELECT *
   FROM F

Resultaat:

fab#	art#	vrd	minvrd
F1	A1	200	150
F1	A2	300	75
F1	A4	150	300
F2	A1	700	200
F2	A2	875	100
F2	A3	0	20
F2	A4	210	210
F2	A5	370	180
F3	A3	15	30
F3	A4	175	100
F4	A4	300	100

7.3.2. Selecteren met attribuutvoorwaarde

Selecteren van (combinatie van) de tupels van de relatie(s) waarvan de waarde van een of meer attributen aan een bepaalde voorwaarde voldoen. De voorwaarde betreft het onderling vergelijken van twee attribuutwaarden of van een attribuutwaarde met een constante.

Voorbeelden

1. De voorraadgegevens van fabriek F3

```
SELECT *
FROM F
WHERE fab# = F3
```

Resultaat:

fab#	art#	vrd	minvrd
F3	A3	15	30
F3	A4	175	100

2. De klanten die artikel A5 hebben besteld.
   Een klant voldoet indien op tenminste één bestelling van hem artikel A5 is besteld.

```
SELECT klant
FROM B, BA
WHERE B.b# = BA.b# AND BA.art# = A5
```

Resultaat:

klant
Snoek
Slager

7.3.3. Selecteren met tupelvoorwaarde

Selecteren van (combinaties van) de tupels van de relatie(s) waarvan de waarde van een of meer attributen al dan niet een tupel is in de in de voorwaarde gespecificeerde relatie.

Voorbeelden.

1. De fabrieken en de door hen te produceren artikelen, waarbij het betreffende artikel ook door fabriek F3 kan worden geproduceerd.
   Een combinatie (fabriek, artikel) voldoet indien het artikel behoort tot de verzameling van artikelen die door fabriek F3 worden geproduceerd.

```
SELECT fab#, art#
FROM F
WHERE art# IN SELECT art#
              FROM F
              WHERE fab# = F3
```

   Van elk tupel van F wordt nagegaan of het art# voorkomt in de relatie die gevormd wordt door de tweede selectopdracht. Deze laatste relatie is:

art#
A3
A4

Zodoende worden de tupels van F geselecteerd waarvan het artikel A3 of A4 is.

Resultaat:

fab#	art#
F1	A4
F2	A3
F2	A4
F3	A3
F3	A4
F4	A4

Dit resultaat bevat tupels die geen nieuwe informatie bevatten, nl. de door F3 te produceren artikelen. Een resultaat zonder deze tupels wordt verkregen met:

```
SELECT fab#, art#
FROM F
WHERE fab# ≠ F3
 AND art# IN SELECT art#
             FROM F
             WHERE fab# = F3
```

2. De fabrieken die artikel A5 niet produceren.

Allereerst merken we op dat de volgende selectopdracht niet het goede resultaat geeft:

```
SELECT fab#
FROM F
WHERE art# ≠ A5
```

Hiermee worden de fabrieken verkregen die tenminste één ander artikel dan A5 produceren.

De selectopdracht waarmee wel het goede resultaat wordt verkregen volgt uit de volgende redenering:
Een fabriek voldoet indien artikel A5 niet behoort tot de verzameling artikelen die hij produceert.
Dit geeft de opdracht:

```
SELECT fab#
FROM F : FX
WHERE A5 NOT IN SELECT art#
                FROM F
                WHERE FX.fab# = F.fab#
```

Van elk tupel van F wordt nagegaan of A5 voorkomt in de relatie die gevormd wordt door de tweede selectopdracht. In tegenstelling tot het voorafgaande voorbeeld is die relatie nu niet vast, maar afhankelijk van het tupel van F dat in beschouwing wordt genomen.
Voor b.v. de tupels van FX met fab# = F1 is het de relatie:

art#
A1
A2
A4

Daar artikel A5 hierin niet voorkomt behoort fabriek F1 tot het resultaat. Op dezelfde wijze worden relaties gevormd behorende bij de tupels van F met

andere fab#'s. Het gevraagde resultaat wordt tenslotte:

fab#
F1
F3
F4

Hetzelfde resultaat wordt ook verkregen met de redenering: Een fabriek voldoet indien hij niet behoort tot de verzameling van fabrieken die artikel A5 wel produceren.
Dit geeft de opdracht:

```
SELECT fab#
FROM F
WHERE fab# NOT IN SELECT fab#
                  FROM F
                  WHERE art# = A5
```

Zonder hier verder op in te gaan wijzen we erop dat de formulering van de vraagstelling van invloed kan zijn op de snelheid van verwerking. Ook de implementatie van SQL speelt hierbij een rol.

7.3.4. Selecteren met relatievoorwaarde

Selecteren van (combinaties van) de tupels van de relatie(s) waarvoor de relatie genoemd in de relatievoorwaarde al dan niet tupels bevat.

Voorbeelden.

1. De bestellingen waarop meer dan één artikel is besteld.
   Een bestelling voldoet indien de verzameling van overige artikelen besteld op die bestelling niet leeg is.
   Hieruit volgt de opdracht:

```
SELECT b#
FROM BA : BAX
WHERE EXISTS SELECT art#
             FROM BA : BAY
             WHERE BAX.art# ≠ BAY.art#
             AND BAX.b# = BAY.b#
```

Met de tweede selectopdracht wordt een relatie verkregen die de overige artikelen van de benaderde bestelling bevat.
Voor het tupel van BAX met b# = B1 en art# = A4 is het de relatie

art#

Daar deze relatie geen tupels bevat behoort bestelling B1 niet tot het resultaat.

Voor het tupel van BAX met b# = B2 en art# = A3 is het de relatie:

art#
A4

Daar deze relatie wel tupels bevat hoort bestelling B2 tot het resultaat.
Op deze manier worden alle tupels van BAX bekeken.

Resultaat:

b#
B2
B3
B4
B5
B6

2. De bestellingen waarop alle te produceren artikelen zijn besteld.
De selectopdracht voor dit soort vragen wordt altijd met de volgende redenering verkregen.
Steeds zijn er twee verzamelingen X en Y te onderscheiden. Bij deze vraag zijn dit de verzameling van alle bestelde artikelen van een bestelling (X) en de verzameling van alle te produceren artikelen (Y). Een bestelling behoort tot het resultaat indien de verzameling van bestelde artikelen de verzameling van alle te produceren artikelen omvat (in dit geval hoogstens gelijk), met andere woorden $X \supset Y$. Hieraan wordt voldaan als voor ieder element van Y geldt dat het tevens een element van X is:

$\forall\, y(y \in Y): y \in X$

en dit is gelijk aan

$\neg\, \exists\, y(y \in Y): (y \notin X)$,

d.w.z. er is geen element van Y dat niet in X voorkomt. Anders gezegd: de verzameling van elementen van Y die niet in X voorkomen is leeg.
Voor deze vraag houdt dit in dat een bestelling voldoet indien de verzameling van die te produceren artikelen die niet op die bestelling voorkomen leeg is.

De verzameling van de te produceren artikelen (Y) wordt verkregen met:

```
SELECT art#
FROM F
```

De verzameling van de bestelde artikelen van een bestelling wordt verkregen met

```
SELECT art#
FROM BA
WHERE BA.b# = de betreffende bestelling.
```

De selectopdracht waarmee de gevraagde bestellingen worden verkregen luidt dus:

```
SELECT b#
FROM B
WHERE NOT EXISTS SELECT art#
                 FROM F
                 WHERE art# NOT IN SELECT art#
                                   FROM BA
                                   WHERE BA.b# = B.b#
```

Voor het tupel van B met b# = B1 zijn de bestelde artikelen

art#
A4

Met

```
SELECT art#
FROM F
WHERE art# NOT IN SELECT art#
                  FROM BA
                  WHERE BA.b# = B.b#
```

wordt dan de volgende relatie gevormd:

art#
A1
A2
A3
A5

Daar deze relatie niet leeg is behoort bestelling B1 niet tot het resultaat.

Voor het tupel van B met b# = B3 zijn de bestelde artikelen:

art#
A1
A2
A3
A4
A5

Met

```
SELECT art#
FROM F
WHERE art# NOT IN SELECT art#
                  FROM BA
                  WHERE BA.b# = B.b#
```

wordt dan de volgende relatie verkregen:

art#

Daar deze relatie leeg is behoort de bestelling B3 tot het resultaat. Op deze manier worden alle tupels van B bekeken.
Resultaat:

b#
B3

3. De bestellingen en bijbehorende klanten van die bestellingen waarop tenminste de artikelen zijn besteld die ook op bestelling B2 zijn besteld.
   De verzameling X is in dit geval de verzameling van bestelde artikelen van een bestelling. De verzameling Y is de verzameling van bestelde artikelen van bestelling B2.
   De verzameling Y wordt verkregen met:

```
SELECT art#
FROM BA
WHERE b# = B2
```

De verzameling X wordt verkregen met:

```
SELECT art#
FROM BA
WHERE b# = betreffende bestelling.
```

De gevraagde klanten worden dus verkregen met

```
SELECT b#, klant
FROM B
WHERE NOT EXISTS SELECT art#
                 FROM BA
                 WHERE b# = B2
                 AND art# NOT IN SELECT art#
                                 FROM BA
                                 WHERE BA.b# = B.b#
```

Resultaat:

b#	klant
B2	Slager
B3	Snoek

7.4. Vraagstukken

Geef de opdrachten in SQL voor het verkrijgen van de volgende gegevens uit de in 6.2. beschreven database:

F(fab#,art#,vrd,minvrd)

B(b#,klant,fab#,datum)

BA(b#,art#,hoev,prijs)

1. De klanten en de bestelnummers van bestellingen die door de klant voor 1-8-1983 bij fabriek F1 zijn geplaatst.
2. De klanten die bij fabriek F1 of fabriek F4 een bestelling hebben geplaatst.
3. De klanten die bij fabriek F2 en fabriek F3 een bestelling hebben geplaatst.
4. De artikelen die door klant Smit zijn besteld, de fabriek waarbij en de datum waarop het artikel is besteld.
5. De klanten die artikel A2 voor minder dan fl. 2,-- hebben besteld.
6. De klanten die artikel A5 niet hebben besteld.
7. De fabrieken die artikel A2 èn artikel A3 niet produceren.
8. De combinaties van twee klanten die op dezelfde dag bij dezelfde fabriek een bestelling hebben geplaatst.
9. De klanten die bij tenminste twee fabrieken een bestelling hebben geplaatst.
10. De fabrieken en de artikelen waarbij het betreffende artikel door meer dan één fabriek wordt geproduceerd.
11. De artikelen die op alle bestellingen van dat artikel voor dezelfde prijs zijn besteld.
12. De artikelen die door alle fabrieken worden geproduceerd.
13. De overige fabrieken die ten minste alle door fabriek F1 te produceren artikelen kunnen produceren.
14. De overige klanten die tenminste de door klant Staal bestelde artikelen hebben besteld.

7.5. Functies in SELECT-opdrachten

SQL zover het tot nu toe is beschreven, is wel krachtig maar ontoereikend voor menig praktisch probleem. Voorbeelden hiervan zijn de vragen naar de totale voorraad van een artikel of het aantal artikelen besteld op een bestelling.

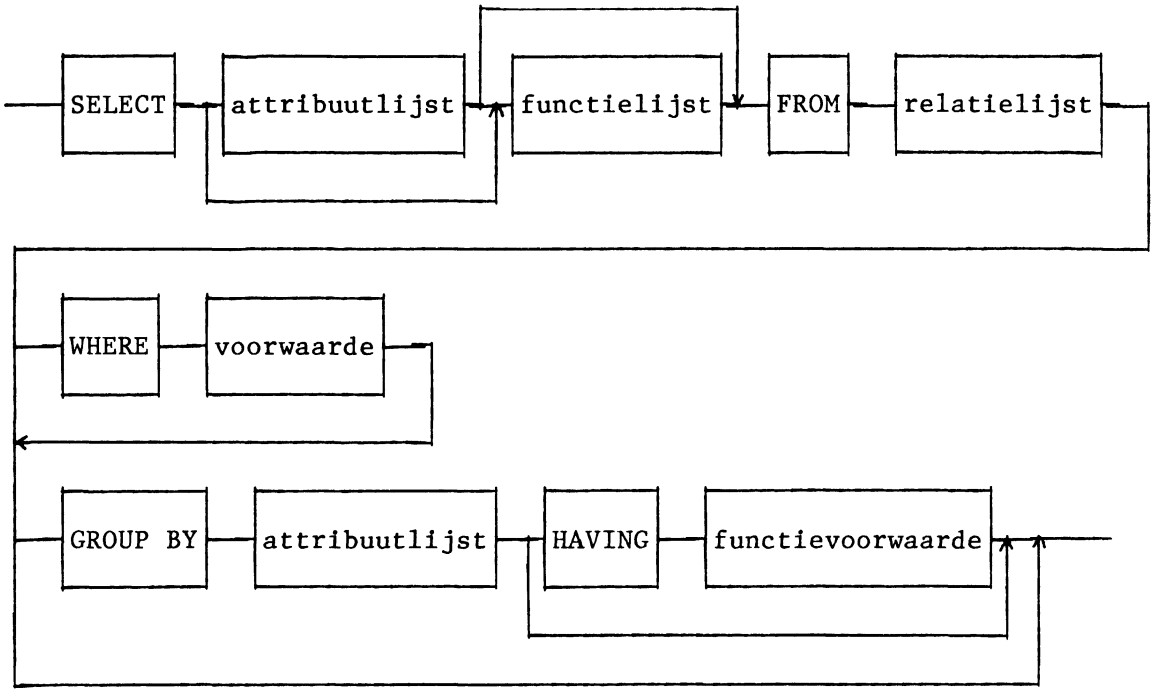
Voor de beantwoording van dit soort vragen beschikt SQL over de volgende functies.

COUNT(attribuut)	: aantal tupels met verschillende attribuutwaarden in (een deel van) een relatie
COUNT(*)	: aantal tupels in (een deel van) een relatie
SUM(attribuut)	: som van de attribuutwaarden in (een deel van) een relatie
AVG(attribuut)	: gemiddelde van de attribuutwaarden in (een deel van) een relatie
MAX(attribuut)	: grootste van de attribuutwaarden in (een deel van) een relatie
MIN(attribuut)	: kleinste van de attribuutwaarden in (een deel van) een relatie

De functies SUM en AVG kunnen alleen worden toegepast op een attribuut waarvan de waarde tot het type geheel of gebroken getal behoort.

De functie COUNT geeft de waarde 0 indien (het deel van) de relatie geen tupels bevat. De overige functies geven dan een ongedefinieerde waarde.

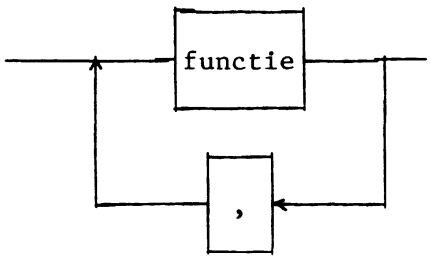
Het syntaxisdiagram van de SELECT-opdracht waarin gebruik wordt gemaakt van functies luidt:



Indien na SELECT geen attribuutlijst volgt dan bevat het resultaat één (combinatie van) waarde(n) voor de genoemde functie(s). Indien na SELECT een attribuutlijst en een functielijst volgen dan bevat het resultaat tupels waaraan per tupel voor iedere functie één waarde is toegevoegd.

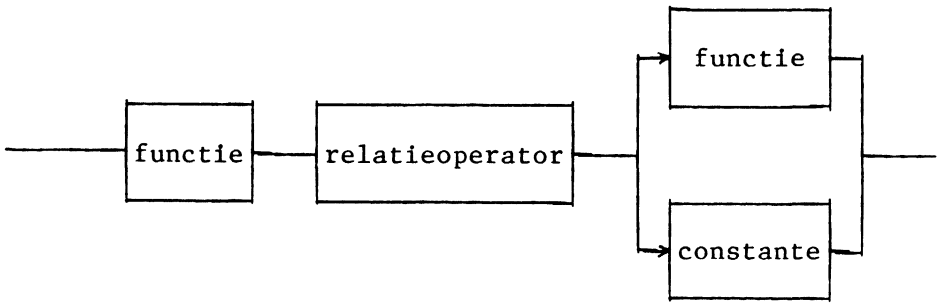
De functielijst wordt gedefinieerd door het diagram:

functielijst



Hierin is functie één van de hiervoor genoemde functies, inclusief attribuut of *. Het syntaxisdiagram van de functievoorwaarde luidt:

functievoorwaarde



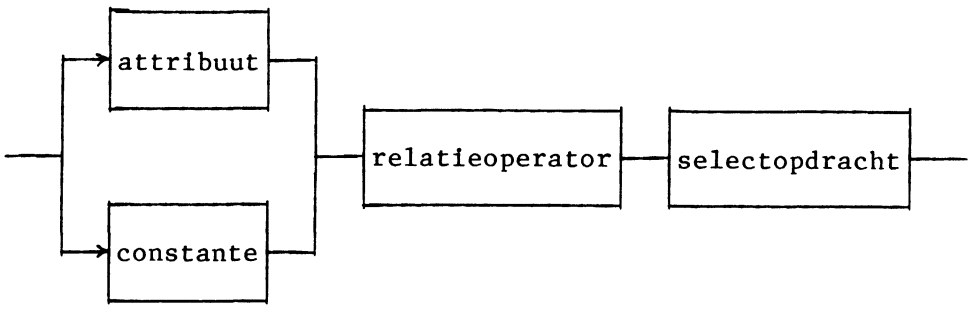
Hierin is relatieoperator één van de operatoren =, <, ≤, > of ≥.

Door gebruik van GROUP BY worden de tupels van de relatie (waaruit wordt geselecteerd) gegroepeerd. De tupels die een zelfde waarde hebben voor de in de GROUP BY genoemde attributen vormen een groep. In dat geval wordt de functie uit de functielijst toegepast op iedere groep. Als geen GROUP BY is opgegeven dan wordt de functie uit de functielijst toegepast op de gehele relatie.

Zoals uit de syntaxis blijkt kan HAVING uitsluitend worden gebruikt in combinatie met GROUP BY. Een functievoorwaarde is dan ook een voorwaarde die a a n een groep wordt gesteld. Alle tupels uit een groep worden geselecteerd indien de groep als geheel voldoet aan de functievoorwaarde.

In de voorwaarde na WHERE kan de volgende bijzondere attribuutvoorwaarde worden gebruikt:

bijzondere attribuutvoorwaarde



Hierin kan selectopdracht uitsluitend een SELECT van één functiewaarde zijn. Alleen dan is er sprake van een zinvolle vergelijking.

7.6. Voorbeelden

De voorbeelden in deze paragraaf hebben betrekking op de relationele database gegeven in paragraaf 6.2. Het relationele schema van deze database is:
F(fab#, art#, vrd, minvrd)
B(b#, klant, fab#, datum)
BA(b#, art, hoev, prijs)

7.6.1. Selecteren van een of meer functiewaarden

Selecteren van een waarde voor een of meer functies. Dit kan alleen met een SELECT opdracht waarin geen GROUP BY voorkomt.

1. Het aantal bestellingen bij fabriek F2.

```
SELECT COUNT(*)
FROM B
WHERE fab# = F2
```

Resultaat:

3

2. De totale hoeveelheid die van artikel A3 is besteld.

```
SELECT SUM(hoev)
FROM BA
WHERE art# = A3
```

Resultaat:

450

7.6.2. Selecteren met gebruik van GROUP BY zonder HAVING

Selecteren van (combinaties van) de tupels van alle groepen van de relatie(s) en bijbehorende waarden voor een of meer functies.

1. Van ieder artikel de totale voorraad.

   SELECT art#, SUM(vrd)
   FROM F
   GROUP BY art#
   Resultaat:

art#	
A1	900
A2	1175
A4	835
A3	15
A5	370

2. Van iedere fabriek het aantal klanten dat na 1-5-1983 tenminste één bestelling bij de fabriek heeft gedaan.

   SELECT fab#, COUNT(klant)
   FROM B
   WHERE datum > 1-5-1983
   GROUP BY fab#
   Resultaat:

fab#	
F1	1
F2	3
F3	1
F4	0

7.6.3. Selecteren met gebruik van GROUP BY en HAVING

Selecteren van (combinaties van) de tupels van die groepen van de relatie(s) en eventuele bijbehorende functiewaarden, waarvan de groep aan een bepaalde functievoorwaarde voldoet. De functievoorwaarde betreft het vergelijken van twee functiewaarden of van een functiewaarde met een constante.

1. De artikelen waarvan in totaal meer dan 500 eenheden zijn besteld, de bestelde hoeveelheid en het aantal bestellingen.

```
SELECT art#, SUM(hoev), COUNT(*)
FROM BA
GROUP BY art#
HAVING SUM(hoev) > 500
```

Resultaat:

art#		
A1	800	3
A2	700	3
A5	1400	2

2. De klanten die voor 1-11-1983 tenminste twee bestellingen hebben gedaan.

```
SELECT klant
FROM B
WHERE datum < 1-11-1983
GROUP BY klant
HAVING COUNT(*) >1
```

Resultaat:

klant
Smit

7.6.4. Selecteren met gebruik van een bijzondere attribuutvoorwaarde

Selecteren van (combinatie van) de tupels van de relatie(s) die aan een bepaalde bijzondere attribuutvoorwaarde voldoen. De voorwaarde betreft het vergelijken van een functiewaarde met een waarde van een attribuut of met een constante.

Voorbeelden

1. De grootste voorraad van artikel A4 en de fabrieken waar deze voorraad van artikel A4 aanwezig is.

```
SELECT fab#, vrd
FROM F
WHERE art# = A4
 AND vrd = SELECT MAX(vrd)
           FROM F
           WHERE art# = A4
```

Resultaat:

fab#	vrd
F4	300

2. De klant die van artikel A5 op één bestelling de grootste hoeveelheid heeft besteld.

```
SELECT klant
FROM B, BA
WHERE B.b# = BA.b#
  AND BA.art# = A5
  AND BA.hoev = SELECT MAX(hoev)
                FROM BA
                WHERE art# = A5
```

Resultaat:

klant
Snoek

3. De klanten die een bestelling van tenminste drie artikelen hebben geplaatst.

```
SELECT klant
FROM B
WHERE 2 < SELECT COUNT(*)
          FROM BA
          WHERE B.b# = BA.b#
          GROUP BY b#
```

7.7. Vraagstukken

1. **Fabrieken**

Geef de opdrachten in SQL voor het verkrijgen van de volgende gegevens uit de in 7.3 beschreven database:

F(fab#,art#,vrd,minvrd)
B(b#,klant,fab#,datum)
BA(b#,art#,hoev,prijs)

1. De artikelen en het aantal verschillende prijzen waarvoor ze besteld zijn.
2. De artikelen en de gemiddelde voorraad van de artikelen waarvan de voorraad kleiner is dan de minimumvoorraad.
3. De klanten, de artikelen en het aantal malen dat de klant het artikel heeft besteld.
4. De artikelen die door slechts één fabriek worden geproduceerd.
5. De artikelen waarvan twee of meer bestellingen van meer dan 200 eenheden zijn.
6. De klanten en de artikelen waarvan door de klant meer dan 400 eenheden zijn besteld.
7. De bestellingen waarop artikel A4 niet voor de laagste prijs is besteld.
8. De fabrieken en de artikelen waarvan de fabriek een voorraad heeft kleiner dan de totale bij de fabriek bestelde hoeveelheid.
9. De klanten die artikel A2 tenminste drie keer hebben besteld.
10. De artikelen en de totaal bestelde hoeveelheid van de artikelen die tenminste drie keer zijn besteld.

2. Makelaars

De bond van makelaars besluit ten behoeve van de koop en verkoop van huizen een database te gaan gebruiken. Van makelaars wordt de kantoornaam (uniek), het kantooradres, de naam van de makelaar en het telefoonnummer opgenomen. Makelaars ontvangen van eigenaars van huizen gedateerde opdrachten tot verkoop. Een eigenaar kan één of meer andere makelaars opdracht tot verkoop geven indien de verkoop van het huis te lang gaat duren. Ook kan dan de vraagprijs worden verlaagd.

Kopers kunnen een bod uitbrengen op een huis bij één van de verkopende makelaars. Zij kunnen dit bod zelf uitbrengen of dat laten doen via de door hen ingeschakelde makelaar. Ieder bod wordt in de database opgenomen. De verdeling van de courtage gebeurt aan de hand van de kosten die de makelaars hebben gemaakt.

Voor deze database zijn de volgende relaties in BCNF ontworpen:

makelaar(kantoornaam, kantooradres, naam, telefoon)
huis(huisadres, eigenaar, vraagprijs)
verkoopopdracht(kantoornaam, huisadres, datum, kosten)
koopopdracht(koper, kantoornaam, datum)
bod(kantoornaam, huisadres, koper, datum, bedrag)

Geef de opdrachten in SQL voor het verkrijgen van de volgende gegevens:

1. De namen en telefoonnummers van de makelaars waaraan na 1-4-1983 huizen in verkoop zijn gegeven met een vraagprijs beneden de fl. 200.000,--.

2. De namen en telefoonnummers van de makelaars waarbij alle huizen van eigenaar Huisboer te koop worden aangeboden.

3. Het aantal huizen waarop na 1-4-1983 tenminste driemaal een bod is uitgebracht.

3. Opleidingsinstituut

Een opleidingsinstituut heeft een programma bestaande uit verschillende modulen. Voor de planning van kursussen wordt een database gebruikt. Voor het geven van een kursus zijn in principe de docenten van de betreffende module beschikbaar. Voor een kursus kunnen zich kursisten aanmelden tot een zeker maximum aantal. Een kursus gaat niet door indien het aantal aanmeldingen kleiner is dan een zeker minimum. Voor iedere kursus wordt een docent gezocht.
Voor deze database zijn de volgende relaties in BCNF ontworpen:

module	M(mod#, modnaam, minimum)
kursus	K(kursus#, mod#, begindatum, plaatsnaam, docent#, maximum)
docent	D(docent#, docentnaam, adres, woonplaats)
inzetbaar	I(docent#, mod#, geschiktheid)
aanmelding	A(kursus#, naam, adres, woonplaats)

1. Geef de opdrachten in SQL voor het verkrijgen van de naam- en adresgegevens van de kursisten die zich voor zoveel kursussen hebben aangemeld dat daaruit blijkt dat ze alle modulen willen gaan volgen.

2. Geef de opdrachten in SQL voor het verkrijgen van de naam- en adresgegevens van de docenten die precies één kursus in Amsterdam verzorgen.

3. Geef de opdrachten in SQL voor het verkrijgen van de kursus#'s van de kursussen die vóór 1-1-1985 beginnen en waarvoor het aantal aanmeldingen kleiner is dan het minimum.

4. Geef de opdrachten in SQL voor het verkrijgen van de namen van de overige docenten die voor alle kursussen van de docent met nummer 23 kunnen worden ingezet.

4. Ziekenhuis

Een ziekenhuis gebruikt voor de administratie van poliklinische behandeling van patienten een database. Het ziekenhuis registreert van patienten NAW-gegevens, geboortedatum en geslacht. Een patient komt onder behandeling van één of meer specialisten. Iedere specialist heeft zijn eigen specialisme. Gedurende de periode van behandeling kan de specialist verschillende onderzoeken voorschrijven. De verschillende mogelijke onderzoeken (b.v. ECG, darmfoto, bloedonderzoek, etc.) zijn gecodeerd en voorzien van een naam. Een patient krijgt voordat een onderzoek wordt uitgevoerd een beschrijving ervan.
Tijdens een behandeling kunnen medicijnen worden voorgeschreven. Voor ieder medicijn afzonderlijk wordt een recept verstrekt waarop de dosis is vermeld. De data van de tijdens de behandeling verrichte onderzoeken en verstrekte recepten moeten kunnen worden opgevraagd.
Het is mogelijk dat een specialist voor een behandeling één collega inschakelt als vervanger.

Voor deze database zijn de volgende relaties in BCNF ontworpen:

specialist	S(S#, naam, specialisme)
patient	P(P#, naam, geb.datum, geslacht, adres, woonplaats)
behandeling	B(S#, P#, begindatum)
type onderzoek	O(O#, naam, beschrijving)
uitgev. onderz.	UO(S#, P#, O#, datum)
recept	R(S#, P#, naam medicijn, datum, dosis)
vervanging	V(S#, P#, VS#)

1. Geef de opdrachten in SQL voor het verkrijgen van de namen en geb.data van de patienten waarbij na 1-7-1984 in opdracht van een internist een nieronderzoek is verricht.

2. Geef de opdrachten in SQL voor het verkrijgen van de namen van de overige patienten die minstens dezelfde onderzoeken hebben ondergaan als patient Bos (geb.dat 21-01-37) heeft ondergaan.

3. Geef de opdrachten in SQL voor het verkrijgen van de namen van de specialisten die voor een behandeling meer dan 5 recepten hebben uitgeschreven.

8. HET NETWERKMODEL

8.1. Inleiding

In databases die gebaseerd zijn op het netwerkmodel, wordt de samenhang tussen objecten tot stand gebracht door ze in ketens op te nemen. In databases die gebaseerd zijn op het relationele model, wordt de samenhang tussen objecten tot stand gebracht door in die objecten eenzelfde waarde voor een attribuut op te nemen. De samenhang op objecttypeniveau wordt in het relationele model tot uitdrukking gebracht door twee attributen die beide op hetzelfde domein worden gedefinieerd.

Een voorbeeld hiervan zijn de relaties:

WERKNEMER(p#, ..., afd#, ...)
AFDELING(afd#, ...)

De objecttypen werknemer en afdeling vertonen samenhang omdat in beide objecttypen het attribuut afd# is opgenomen. Dat de werknemer Jansen op de boekhoudafdeling werkt wordt vastgelegd door in beide objecten dezelfde waarde voor afd# op te nemen.

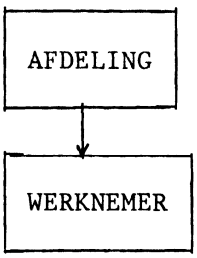
In het netwerkmodel wordt een objecttype een recordtype genoemd. De samenhang op recordtypeniveau wordt tot uitdrukking gebracht door een z.g. setrelatie tussen recordtypen. Er worden vier soorten setrelaties onderscheiden. Hiermee kan een netwerkmodel voor een toepassing worden opgebouwd.

8.2. Settype met twee recordtypen

Aan de hand van de volgende toepassing laten we zien wat onder een settype met twee recordtypen moet worden verstaan.
Een bedrijf bestaat uit afdelingen waarop werknemers werken. Iedere werknemer behoort tot precies één afdeling.

Het netwerkmodel van de database voor deze toepassing wordt ontworpen op grond van de redenering:
1. Op iedere afdeling werken nul, of meer werknemers;

2. Iedere werknemer werkt op één afdeling.
Het netwerkmodel wordt dan:



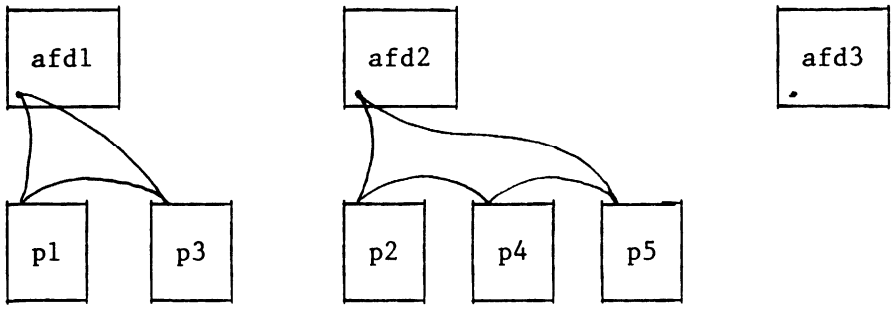
Deze constructie wordt een settype met twee recordtypen genoemd.

Allereerst zien we dat de attributen van de recordtypen niet in het model worden beschreven. Uiteraard heeft het recordtype AFDELING wel eigenschappen zoals afd#, afdnaam, budget, chef, etc. maar die worden pas expliciet gemaakt in de beschrijving van de database (zie hoofdstuk 9).
Ook valt op dat het settype twee recordtypen bevat, die door een pijl met elkaar zijn verbonden. Het recordtype waar de pijl naar wijst (WERKNEMER) wordt het membertype genoemd.
Het recordtype van waaruit de pijl is getrokken (AFDELING), wordt het ownertype genoemd.
Ieder recordtype kan in meer dan één settype worden opgenomen. Een recordtype kan daardoor in nul of meer verschillende settypen een ownertype zijn en tevens in nul of meer verschillende settypen een membertype.

Met dit settype wordt een 1:n relatie tussen twee recordtypen vastgelegd. Bij een owner horen volgens de betekenis van het settype nul of meer members. Bij een member hoort volgens de betekenis van het settype precies één owner. Een objecttype is de abstractie van een verzameling objecten. Zo is een recordtype de abstractie van een verzameling records die in dit model recordoccurrences worden genoemd. Een settype is ook een abstractie van een verzameling waarvan de elementen setoccurrences worden genoemd. Een setoccurrence is het geheel van één of meer recordoccurrences, n.l. 1 owneroccurrence en n ($n \geq 0$) bijbehorende memberoccurrences.

Voorbeeld
Veronderstel dat het bedrijf uit 3 afdelingen (afd1, afd2 en afd3) bestaat en vijf werknemers (p1, p2, p3, p4 en p5) in dienst heeft, waarbij p1 en p3

op afdeling 1 en p2, p4 en p5 op afdeling 2 werken.
De database bevat dan de volgende drie setoccurrences :



De setoccurrence die alleen afdeling 3 bevat heet een lege setoccurrence, omdat de setoccurrence geen memberoccurrences bevat. Een setoccurrence moet wel altijd een owneroccurrence bevatten.
Hierdoor wordt automatisch voldaan aan de referential integrity. Een in de database opgeslagen occurrence van werknemer maakt altijd deel uit van een setoccurrence en kan dus nooit bij een niet bestaande afdeling horen.

Het relationele model voor deze toepassing is:

WERKNEMER(p#, ..., afd#)
AFDELING(afd#, afdnaam, budget, chef)

Op objecttypeniveau vertonen het netwerkmodel en het relationele model dus grote overeenkomsten. Beide kennen in dit geval twee objecttypen met dezelfde betekenis die op één uitzondering na dezelfde attributen hebben (zie paragraaf 8.7.).
De opslag van objecten in de database is echter geheel verschillend. In een netwerk database worden de objecten gegroepeerd in setoccurrences; in een relationele database worden records niet gegroepeerd, maar is de samenhang vastgelegd door gelijke waarden voor afd#.
In de relationele database wordt daardoor niet automatisch voldaan aan de referential integrity omdat het afd# van een werknemer in principe een waarde zou kunnen bevatten die niet voorkomt in één van de tupels van de relatie AFDELING.

Samenvatting

Op settypeniveau geldt:
1. Een settype bestaat uit twee recordtypen, een ownertype en een member-

type.

2. Ieder recordtype kan in meer dan één settype worden opgenomen.

Op setoccurrence niveau geldt:

1. Er zijn net zoveel setoccurrences van een settype als er owneroccurrences zijn.
2. Een setoccurrence bevat precies één owneroccurrence en n (n≥0) memberoccurrences. Een setoccurrence is leeg indien hij geen memberoccurrences bevat.
3. Iedere memberoccurrence maakt deel uit van precies één setoccurrence van een settype.

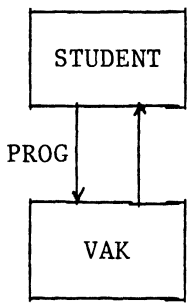
8.3. Samengesteld netwerk

De onderwijsadministratie van een afdeling van de TH legt het studieprogramma van de verschillende studenten vast. Een studieprogramma bevat een aantal vakken.

Op grond van de redenering

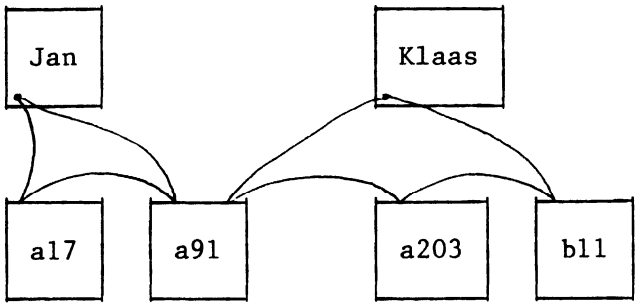
1. Een student heeft een studieprogramma bestaande uit n (n≥0) vakken;
2. Een vak kan in studieprogramma's van n (n≥0) studenten worden opgenomen,

zou het volgende netwerkmodel kunnen worden ontworpen:



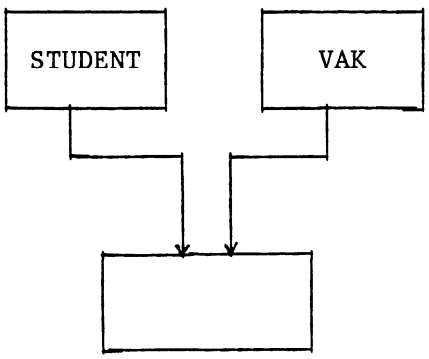
Echter dit ontwerp voldoet niet omdat hieruit af te lezen valt dat een bepaald vak in het studieprogramma van slechts één student kan worden opgenomen. Deze conclusie kunnen we trekken op grond van de eis dat een memberoccurrence deel uitmaakt van precies één setoccurrence.
Aan de hand van een setoccurrence-diagram zullen we dit verduidelijken. Veronderstel dat er twee studenten zijn (Jan en Klaas) en vier vakken (a17, a91, a203, b11). Het studieprogramma van Jan bevat de vakken a17 en a91. Het studieprogramma van Klaas de vakken a91, a203, b11.

De database zou dan de volgende setoccurrences van settype PROG bevatten:



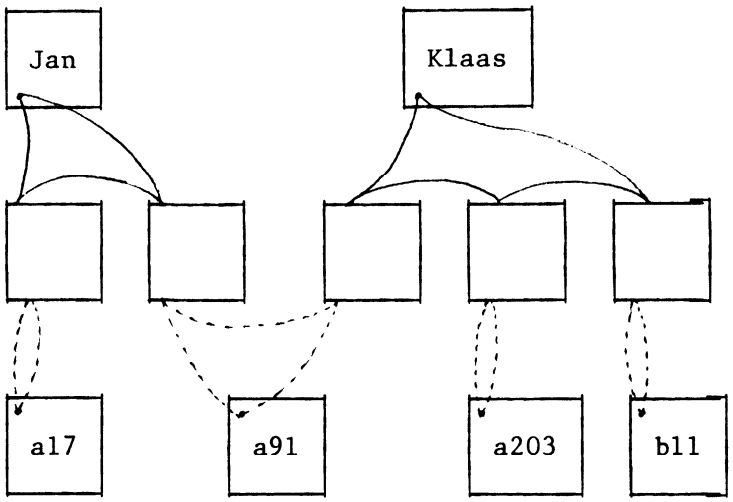
Het vak a91 maakt nu echter deel uit van twee setoccurrences n.l. één waarbij Jan de owner is en één waarbij Klaas de owner is.
Aangezien het niet is toegestaan dat een member deel uitmaakt van meer dan één setoccurrence van hetzelfde type, zullen we een ander ontwerp moeten maken.

Een nadere bestudering van de gegeven redenering leert ons dat er sprake is van een n:m relatie tussen de recordtypen STUDENT en VAK en niet van twee afzonderlijke 1:n relaties.
Een n:m relatie tussen twee recordtypen levert altijd een samengesteld netwerk op:



Het ingevoerde recordtype heeft uiteraard een betekenis. Hier kan het recordtype het beste worden omschreven met PROGRAMMA-ONDERDEEL. Het recordtype behoeft geen attributen te bevatten. Zijn ze wel aanwezig, dan moeten zij betrekking hebben op de combinatie student-vak, b.v. het cijfer van het laatste afgelegde tentamen. Eventueel kunnen ook de sleutelattributen van de recordtypen STUDENT en VAK er in worden opgenomen (zie 8.7.).

De database die ontstaat op grond van het samengestelde netwerk STUDENT-VAK bevat de volgende setoccurrences:



Het relationele model voor deze toepassing is:

VAK(vakcode, vaknaam, zwaarte)

STUDENT(studie#, NAW, studierichting)

PROGRAMMA(studie#, vakcode)

Ook in dit geval is er een grote overeenkomst tussen het netwerkmodel en het relationele model.

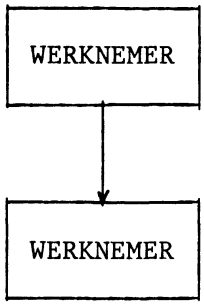
8.4. Settype met één recordtype

Een bedrijf heeft werknemers in dienst. Van deze werknemers zijn verschillende gegevens vastgelegd. Een van deze gegevens is wie de baas is van de werknemer. De baas is ook een werknemer.

Op grond van de redenering:

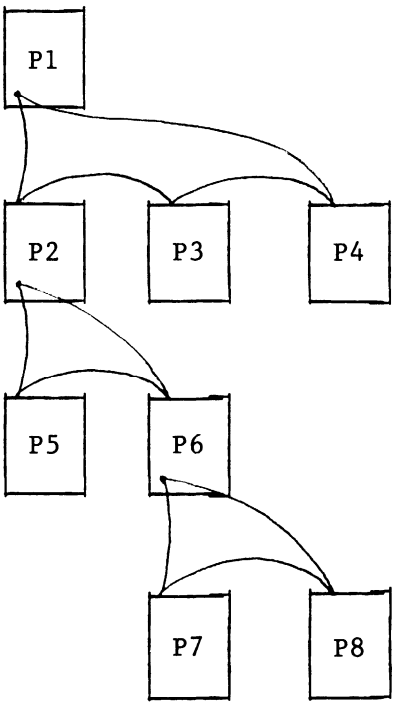
1. Een werknemer geeft aan n ($n \geq 0$) werknemers leiding;
2. Een werknemer krijgt van één werknemer leiding,

zou het volgende netwerkmodel kunnen worden ontworpen:



Echter dit ontwerp voldoet niet omdat dan de gegevens van een werknemer die tevens baas is tweemaal in de database zijn vastgelegd.
In feite bevat de redenering een 1:n relatie van het recordtype WERKNEMER met zichzelf. Zo'n relatie leidt tot een settype met één recordtype:

Veronderstel dat het bedrijf acht werknemers in dienst heeft, p1 t/m p8. Werknemer p1 geeft leiding aan p2, p3, p4; werknemer p2 aan p5 en p6 en werknemer p6 aan p7 en p8.
De database bevat dan de volgende setoccurrences:



Ook hier zien we weer overeenkomst met het relationele model, dat er als volgt uitziet:

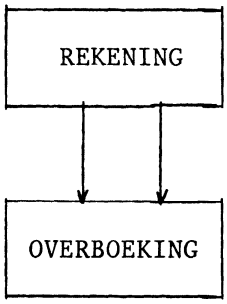
WERKNEMER(p#, NAW, baas)

De foreign key baas wordt in het netwerkmodel dus weergegeven door een set-type.

8.5. Eenvoudig netwerk

De klanten van de girodienst beschikken over rekeningen waarop zij overboekingen van andere rekeningen kunnen ontvangen en betalingen naar andere rekeningen kunnen verrichten. Op verzoek van de boekhouder wordt niet alleen het saldo bijgehouden maar worden tevens de overboekingen geregistreerd.

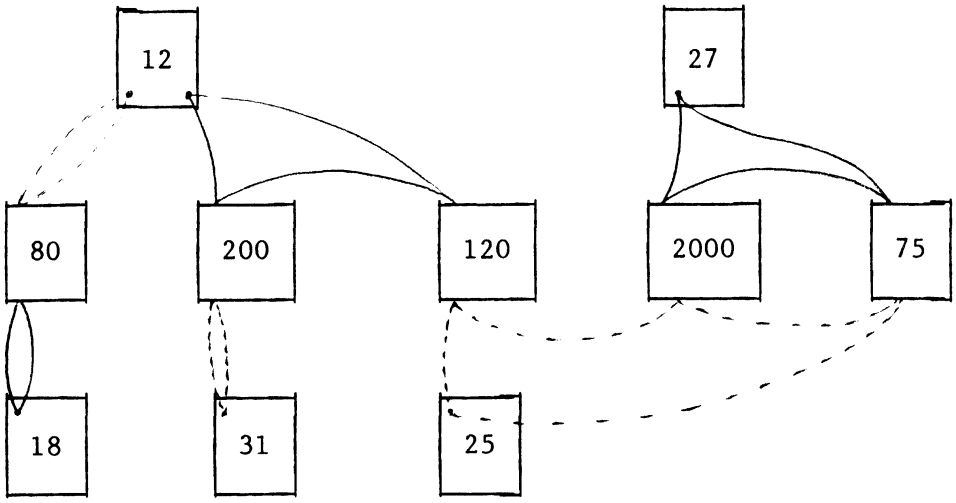
Op grond van de redenering
1. Een rekening ontvangt van n (n≥0) andere rekeningen geld;
2. Een rekening stuurt naar n (n≥0) andere rekeningen geld,
ontstaat het volgende netwerkmodel:



Uit de redenering volgt dat er sprake is van een n:m relatie van het recordtype REKENING met zichzelf en niet van twee afzonderlijke 1:n relaties tussen de recordtypen REKENING en OVERBOEKING.
Een n:m relatie van een recordtype met zichzelf geeft altijd een eenvoudig netwerk.
Het recordtype OVERBOEKING heeft uiteraard een betekenis, die tot uitdrukking komt in de attributen die het heeft (datum, bedrag).

Veronderstel dat de girodienst 5 klanten heeft met de girorekeningen 12, 27, 18, 31 en 25.

Voor deze klanten zijn de volgende overboekingen gedaan:
Van rekening 12 naar rekening 31 op 1-1-1984 het bedrag van fl. 200,--, naar rekening 25 op 2-1-1984 het bedrag van fl. 120,--.
Van rekening 27 naar rekening 25 op 2-1-1984 het bedrag van fl. 2.000,-- en op 13-1-1984 nog eens fl. 75,--.
Van rekening 18 naar rekening 12 op 5-1-1984 het bedrag van fl. 80,--.
De database bevat dan de volgende setoccurrences:



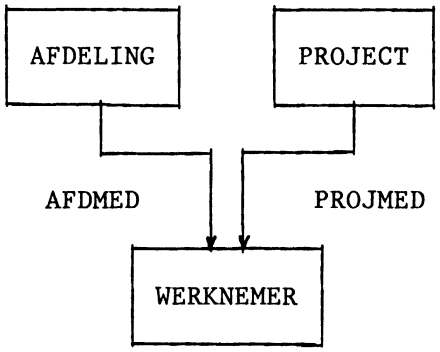
Voor de overzichtelijkheid zijn in de records van het type OVERBOEKING alleen de bedragen vermeld.

8.6. Settype met sometime membership

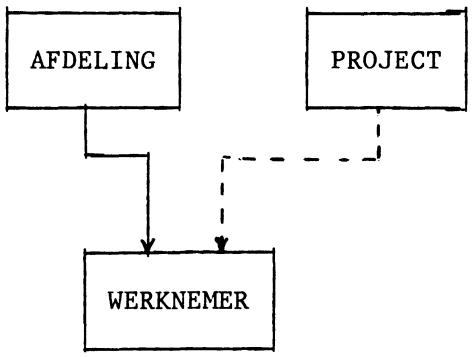
Een bedrijf voert projecten uit. De projecten worden bemand door werknemers van het bedrijf, waarbij een werknemer hoogstens voor één project kan worden ingezet. Niet alle werknemers worden op projecten ingezet. Iedere werknemer behoort tot één afdeling.

Hieruit kunnen we de volgende setrelaties afleiden:
1. Een 1:n relatie tussen de recordtypen AFDELING en WERKNEMER
   Op een afdeling werken n (n≥0) werknemers.
   Iedere werknemer behoort tot één afdeling.
2. Een 1:n relatie tussen de recordtypen PROJECT en WERKNEMER
   Aan een project werken n (n≥0) werknemers.
   Iedere werknemer die aan een project werkt, doet dit voor slechts één project.

Op grond hiervan ontstaat het volgende netwerkmodel:



In de hierbij horende database zullen alle werknemers van het bedrijf worden opgenomen.
Iedere werknemer moet deel uitmaken van een setoccurrence van het settype AFDMED. Vereist is ook dat iedere werknemer deel uitmaakt van een setoccurrence van het settype PROJMED (zie de samenvatting in paragraaf 8.2), hetgeen zou betekenen dat iedere medewerker wordt ingezet op een project. Door gebruik te maken van het settype met sometime membership kan er voor worden gezorgd dat alle werknemers in de database kunnen worden opgenomen en dat niet iedere medewerker wordt ingezet op een project. Zo'n bijzonder settype wordt aangeduid met een gestippelde pijl. Het netwerkmodel van de eerder beschreven toepassing wordt daardoor:



Een gewoon en een sometime settype verschillen alleen t.a.v. de deelnamen van memberoccurrences in setoccurrences. Bij een gewoon settype komt iedere memberoccurrence voor in precies één setoccurrence. Bij een sometime settype komt iedere memberoccurrence voor in hoogstens één setoccurrence.

Een 1:n relatie van een recordtype met zichzelf werd afgebeeld (zie paragraaf 8.4) met het volgende netwerkmodel:

BAAS

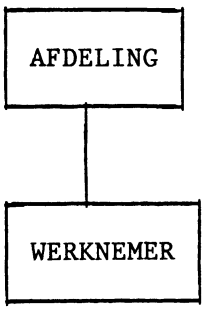
Niet iedere werknemer heeft echter een andere werknemer als baas (b.v. de directeur van het bedrijf) zodat de 1:n relatie van WERKNEMER met zichzelf moet worden afgebeeld met een sometime settype:

BAAS

Eenvoudig is in te zien dat iedere 1:n relatie van een recordtype met zichzelf door een sometime settype moet worden afgebeeld.

8.7. Informatie dragend settype

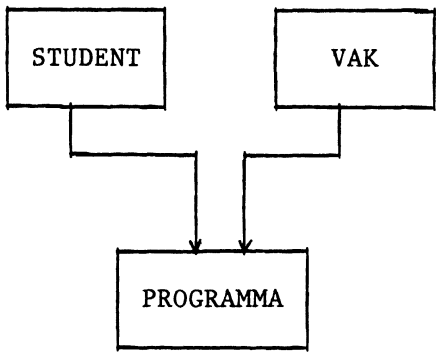
We beschouwen het volgende netwerkmodel.



Indien in het recordtype WERKNEMER het attribuut afd# niet wordt opgenomen, dan wordt gesproken van een informatiedragend (eng.: information bearing) settype, omdat de afdeling van een werknemer alleen wordt vastgelegd door de betreffende werknemersoccurrence op te nemen in de setoccurrence waarvan de bijbehorende afdelingsoccurrence de owner is.
Indien in het recordtype WERKNEMER het attribuut afd# wel wordt opgenomen, dan wordt gesproken van een niet-informatiedragend settype. Het bezwaar hiervan is dat een werknemersoccurrence een waarde voor afd# kan bevatten

die verschilt van de waarde voor afd# in de bijbehorende afdelingsoccurrence. Een voordeel is echter dat het memberrecord "volledig" is, zodat in sommige toepassingen een benadering van het ownerrecord achterwege kan blijven.

In het samengesteld netwerk



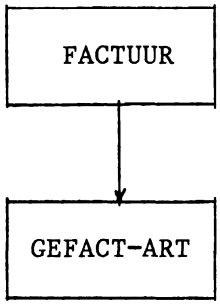
zal het recordtype PROGRAMMA dus geen attributen behoeven te hebben als beide sets informatiedragend zijn.

8.8. Uitgewerkte vraagstukken

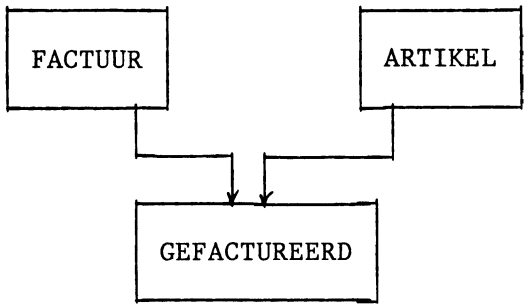
8.8.1. **Niet betaalde facturen**

Een bedrijf legt in een database de gegevens vast van de nog niet betaalde facturen van klanten. De klanten bestellen bij het bedrijf op een bestelling één of meer artikelen tegen een bepaalde prijs. Een bestelling wordt altijd in zijn geheel afgeleverd en voorzien van een gedateerde factuur.
De facturen worden van elkaar onderscheiden door een factuurnummer. Op de factuur worden bestelnummer (uniek), besteldatum, het door de klant te betalen bedrag, en per artikel het aantal en de prijs vermeld. De prijs van een artikel kan per klant verschillen.

Hieruit wordt de volgende setrelatie afgeleid.
Op een factuur worden n artikelen gefactureerd. Een gefactureerd artikel behoort tot één factuur. Een 1:n relatie tussen de recordtypen FACTUUR en GEFACT-ART. Dit geeft de structuur:



Opgemerkt wordt dat op grond van de volgende redenering een onjuiste structuur ontstaat. Op een factuur worden n artikelen gefactureerd. Een artikel kan op n facturen voorkomen. Dus een n:m relatie tussen de recordtypen FACTUUR en ARTIKEL.
Dit geeft de structuur



Deze oplossing zou correct zijn indien het bedrijf ook gegevens over artikelen vastlegt. In het kader van deze toepassing heeft het recordtype ARTIKEL geen attributen, dient ook niet om een relatie vast te leggen en is derhalve geen objecttype.

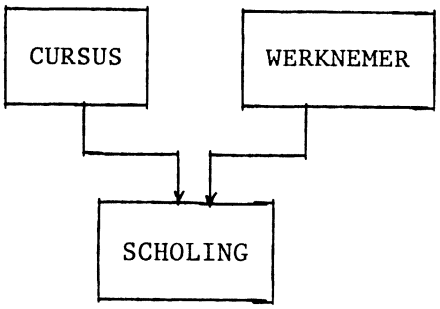
8.8.2. **Scholing**

We beschouwen de volgende toepassing.
Een bedrijf legt in een database gegevens vast over zijn personeelsleden en de cursussen die zij hebben gevolgd.
De personeelsleden worden van elkaar onderscheiden door een personeelsnummer(p#). Verder zijn NAW en geboortedatum van ieder personeelslid van belang. Een personeelslid kan in een jaar een of meer cursussen volgen. Een personeelslid volgt een cursus hoogstens éénmaal. De cursussen worden van elkaar onderscheiden door een cursusnummer (c#) en hebben een naam. In een

cursus wordt de inhoud van een bepaald boek behandeld. Elk jaar kan een ander boek worden gebruikt. Nadat het personeelslid een cursus heeft gevolgd neemt hij/zij een of meer keren deel aan een examen.

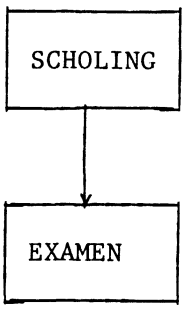
Hieruit worden de volgende setrelaties afgeleid.

1. Een werknemer kan n cursussen volgen. Een cursus kan door n werknemers worden gevolgd. Dit geeft een n:m relatie tussen de recordtypen CURSUS en WERKNEMER, waarbij CURSUS moet worden opgevat als een cursus in een bepaald jaar.
   Hierdoor ontstaat de structuur:

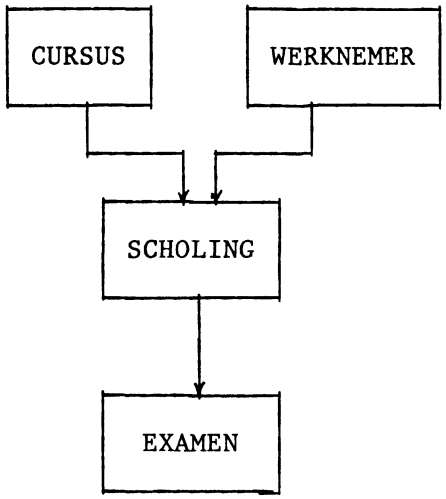


   Als gevolg van deze relatie ontstaat dus het recordtype SCHOLING.

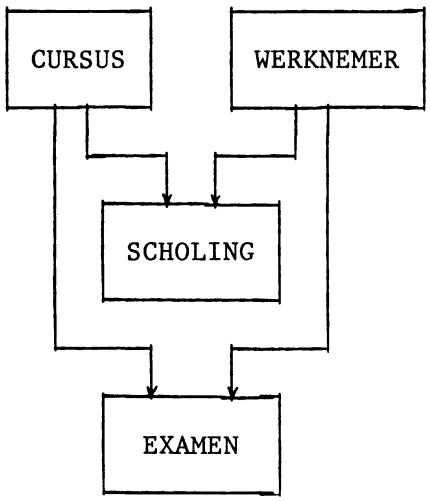
2. Na een scholing kan de werknemer aan n examens van de cursus deelnemen. Een examenresultaat wordt door één werknemer voor een examen van één cursus behaald. Er bestaat dus een 1:n relatie tussen de recordtypen SCHOLING en EXAMEN. Dit geeft de structuur:



Samengevat ontstaat het volgende netwerkmodel:



Een ontsporing die bij de uitwerking van dit vraagstuk gemakkelijk kan worden gemaakt is dat het recordtype EXAMEN niet met het recordtype SCHOLING wordt verbonden maar met de recordtypen CURSUS en WERKNEMER. Dit gebeurt dan op grond van de redenering dat een werknemer aan n examens van een cursus kan deelnemen en dat aan een examen van een cursus n werknemers deelnemen. Hierdoor ontstaat de structuur:



De consequentie hiervan is echter dat een werknemer nu wel aan een examen van een cursus kan deelnemen zonder de cursus gevolgd te hebben. Aangezien in de probleemstelling wordt gesteld dat een personeelslid pas nadat hij de cursus heeft gevolgd kan deelnemen aan examens, is deze oplossing niet correct.

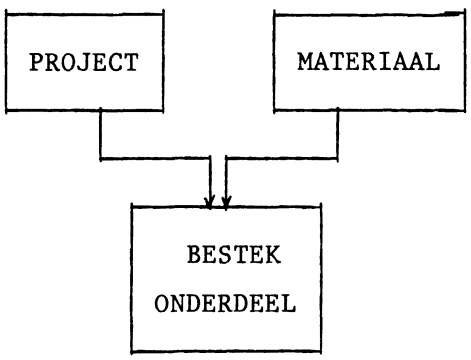
8.8.3. **Bouwonderneming**

Een bouwonderneming ontwerpt een database waarin gegevens over projecten, materialen en leveranciers worden opgenomen. Projecten worden van elkaar onderscheiden door hun naam; materialen en leveranciers door een nummer. Van materialen wordt tevens de naam en van leveranciers de NAW-gegevens opgenomen in de database. Opdrachtgevers van projecten vragen de bouwonderneming op basis van een bestek een offerte uit te brengen. In het bestek van het project is vastgelegd hoeveel van ieder materiaal nodig is. Bij het opstellen van een offerte wordt voor elk materiaal van een calculatieprijs (≡ de huidige gemiddelde prijs) per eenheid uitgegaan. Er wordt uitsluitend met materiaalkosten rekening gehouden. Tevens wordt vastgesteld in hoeveel tijd het project kan worden gerealiseerd.

Van de projecten die worden gerealiseerd worden de gegevens die nodig zijn om na te kunnen gaan of het project winstgevend is geweest, eveneens vastgelegd. De materialen worden geleverd door de leveranciers. Van een leverantie wordt vastgelegd de datum, de hoeveelheden geleverde materialen en de prijs.

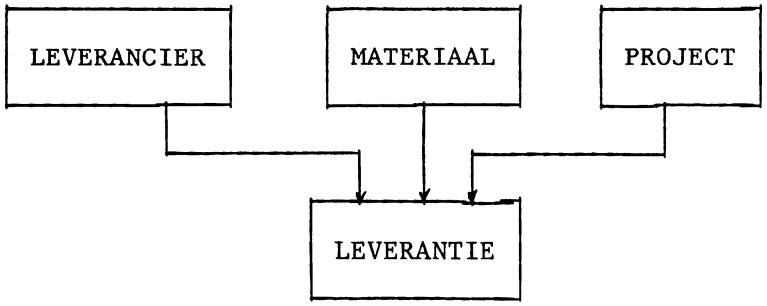
Hieruit leiden we de volgende relaties af:

1. Een project heeft een bestek waarin n (n>0) materialen zijn opgenomen. Een materiaal kan in het bestek van n (n>0) projecten worden opgenomen. Een n:m relatie tussen de recordtypen PROJECT en MATERIAAL.

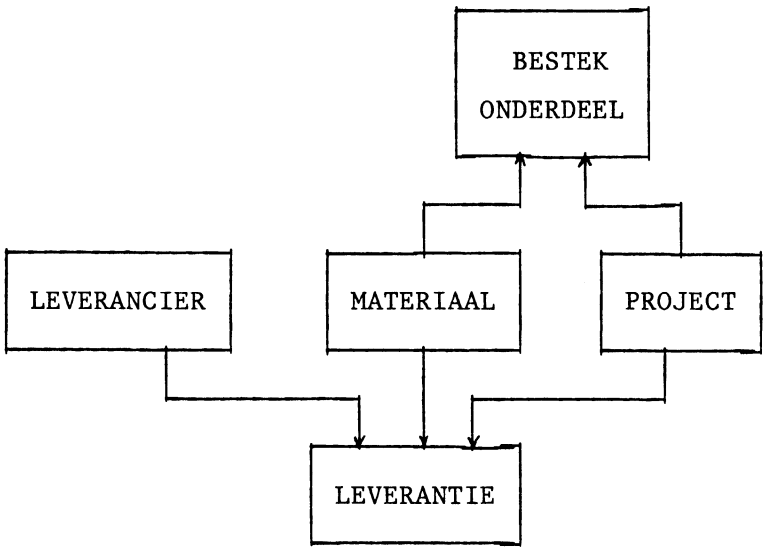


2. Een leverancier levert meer dan één materiaal aan meer dan één project. Een materiaal wordt door meer dan één leverancier aan meer dan één project geleverd. Aan een project wordt door meer dan één leverancier meer

dan één materiaal geleverd. Hiermee is een relatie beschreven tussen de recordtypen LEVERANCIER, MATERIAAL en PROJECT.
Deze relatie kan niet worden afgebeeld met een van de vier basisstructuren. De structuur die hiervoor moet worden gebruikt is:



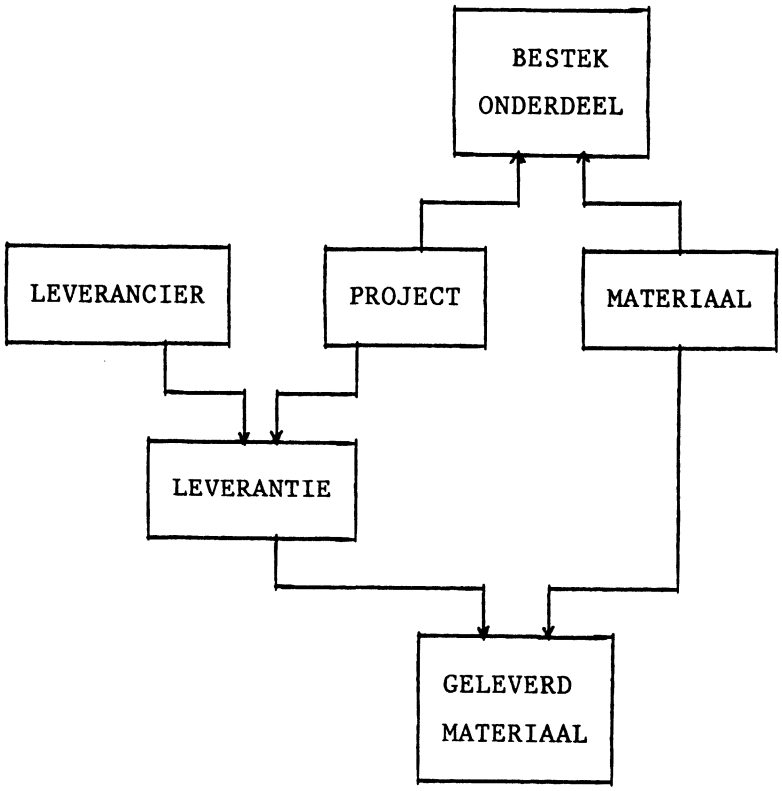
Samengevoegd ontstaat het volgende netwerkmodel:



Een levering in deze context moet dus worden opgevat als de levering van één bepaalde hoeveelheid van één materiaal op een bepaalde dag aan één project door één leverancier.
Meer in overeenstemming met de omschrijving zou een leverantie ook anders kunnen worden opgevat n.l. als de levering van bepaalde hoeveelheden van één of meer materialen op een bepaalde dag aan één project door één leverancier.
Dan ontstaan twee n:m relaties, n.l.:

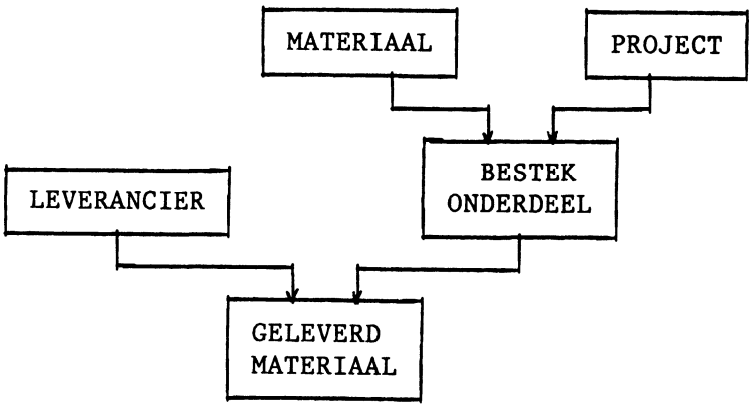
1. Een leverancier kan aan n projecten hebben geleverd. Aan een project kunnen n leveranciers hebben geleverd.
2. Een leverantie betreft n materialen. Een materiaal kan met n leveringen zijn geleverd.

Het netwerk model zou dan worden:



Anders dan bij de voorgaande oplossing bevat het recordtype LEVERANTIE (afgezien van eventueel lev# en proj#, zie 8.7.) nu uitsluitend de datum van de levering.

Het volgende netwerkmodel is niet geschikt voor de geschetste toepassing:



Het idee dat hier achter steekt is dat iedere leverancier een gedeelte van de materialen uit het bestek van een project levert. In de bijbehorende database kan echter de levering van een materiaal dat niet in het bestek voorkomt, niet worden vastgelegd. Een ingrijpende en onnodige beperking die er voor zorgt dat dit netwerkmodel niet geschikt is voor deze toepassing.

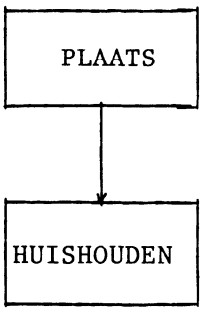
8.8.4. **Bevolking**

Het ministerie van Binnenlandse Zaken ontwerpt een database waarin de gegevens over plaatsen, personen en scholen worden opgeslagen.
Van iedere plaats is het gewenste aantal inwoners vastgesteld. Iedere persoon krijgt een pers# (uniek). Van iedere persoon wordt opgenomen naam, adres, woonplaats, geboortedatum en geslacht.

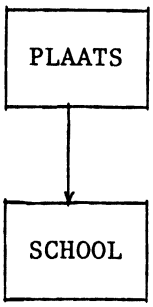
Personen voeren een gemeenschappelijke huishouding. Een van deze personen is het hoofd van het huishouden. Iedere persoon behoort tot één huishouden.
Personen zitten op hoogstens één school.

Scholen worden van elkaar onderscheiden door de naam en de plaats waarin de school is gevestigd.
Een school heeft een vastgesteld budget.

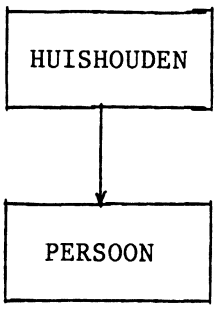
Hieruit leiden we de volgende setrelaties af:
1. In een plaats zijn n huishoudens gevestigd. Een huishouden is in één plaats gevestigd.
   Een 1:n relatie tussen de recordtypen PLAATS en HUISHOUDEN



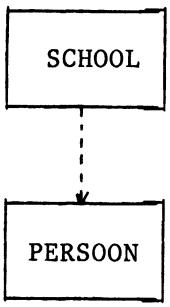
2. In een plaats zijn n scholen gevestigd. Een school is in één plaats gevestigd.
   Dit geeft:



3. Een huishouden bestaat uit n (n≥1) personen. Iedere persoon behoort tot één huishouden. Dit geeft:

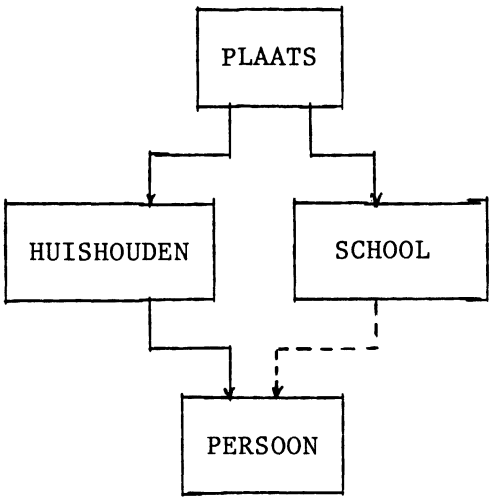


4. Een school heeft n personen als leerling. Een persoon kan op hoogstens één school zitten.
   Dit geeft:



Deze 1:n relatie wordt weergegeven door een settype met sometime membership, omdat niet iedere persoon op een school zit.

Samengevoegd wordt het netwerkmodel voor deze toepassing:

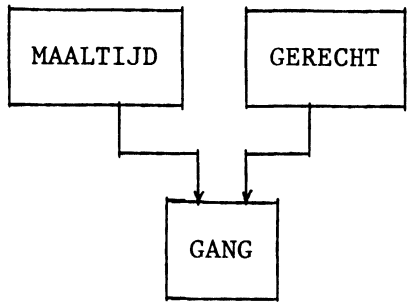


8.8.5. **Kookboek**

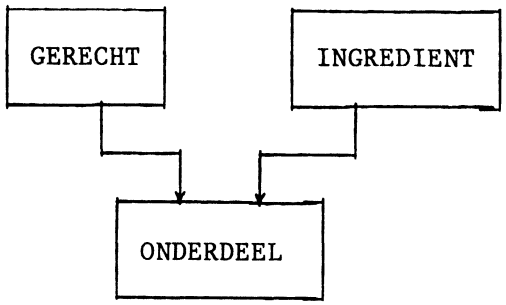
De schrijver van een boek over het samenstellen van maaltijden maakt gebruik van een database.
Maaltijden worden van elkaar onderscheiden door een nummer. Er zijn verschillende soorten maaltijden (b.v. vis, vegetarisch) afkomstig uit diverse landen. Een maaltijd bestaat uit gangen waarin één gerecht (b.v. krabcocktail, tomatensoep, andijviestamppot) wordt geserveerd.
Gerechten worden bereid uit ingrediënten (b.v. aardappelen, bloemkool, zout, biefstuk). Per gerecht wordt aangegeven hoeveel van een ingrediënt nodig is voor een maaltijd voor twee personen. Indien een ingrediënt in een bepaald gerecht niet beschikbaar is, wordt aangegeven welke andere ingrediënten als vervanging kunnen dienen en welke hoeveelheid dan nodig is.

Hieruit leiden we de volgende setrelaties af.

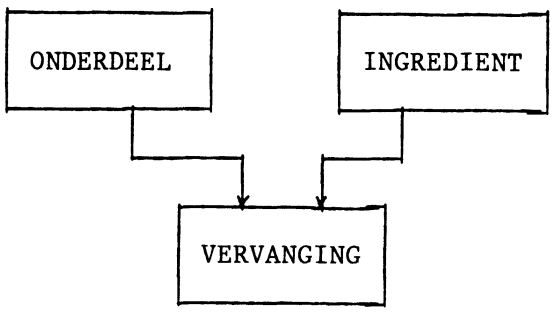
1. Een maaltijd bestaat uit n gangen waarin een gerecht wordt geserveerd.
   Een gerecht kan in verschillende maaltijden worden geserveerd.
   Een n:m relatie tussen de recordtypen MAALTIJD en GERECHT



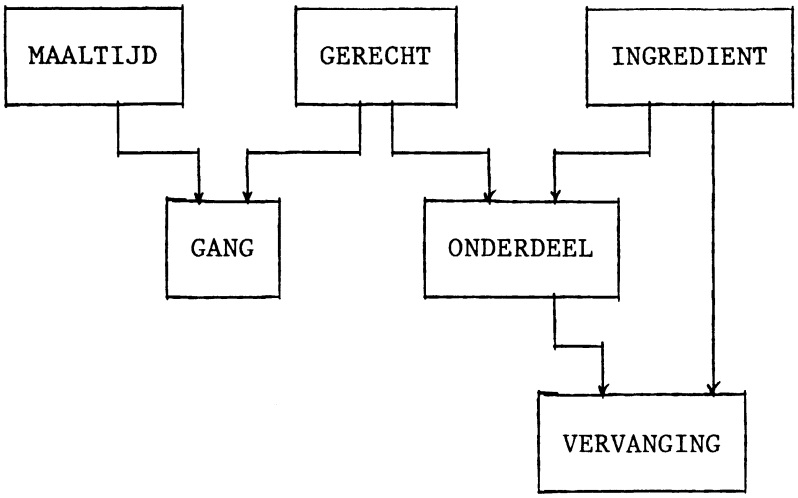
2. Een gerecht wordt bereid uit n ingrediënten. Een ingrediënt kan in meer dan één gerecht worden gebruikt.
   Een n:m relatie tussen de recordtypen GERECHT en INGREDIENT



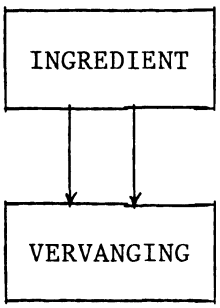
3. Een ingrediënt in een gerecht kan door n ingrediënten worden vervangen. Een ingrediënt kan voor n ingrediënten in bepaalde gerechten als vervanger dienen.
   Een n:m relatie tussen de recordtypen INGREDIENT en ONDERDEEL



Samengevoegd ontstaat het volgende netwerkmodel

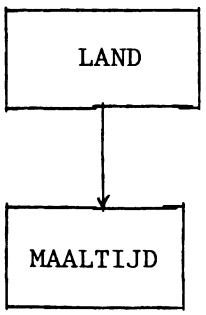


Opgemerkt wordt dat met de structuur



de toepassing niet goed wordt weergegeven, daar een ingredient nu onafhankelijk van het gerecht altijd door dezelfde ingredienten kan worden vervangen. Echter in het ene gerecht kan b.v. rijst wel door macaroni worden vervangen, in het andere niet.

Op grond van de redenering dat uit een land meer dan een maaltijd afkomstig is zou de volgende structuur kunnen ontstaan:



Omdat we niet in gegevens over landen zijn geïnteresseerd en omdat het recordtype niet nodig is om een n:m relatie vast te leggen, is LAND geen recordtype doch een attribuut van het recordtype MAALTIJD.
Het feit dat het recordtype LAND geen attributen heeft is hiervoor een aanwijzing. Echter, de recordtypen GERECHT en INGREDIENT bezitten beide ook geen attributen. Toch worden deze wel als recordtypen opgevoerd omdat we over gerechten en ingrediënten uiteraard wel gegevens willen vastleggen en omdat anders de samenhang tussen maaltijden, gerechten, ingrediënten en vervangingen niet in de structuur kan worden aangegeven.

8.9. Vraagstukken

1. **Onderwijs**

Het ministerie van Onderwijs laat een database ontwerpen ten behoeve van de verdeling van de beschikbare middelen voor het middelbare dagonderwijs.
Van belang hiervoor zijn de gegevens:
- de namen van de scholen;
- in welke gemeente een school is gevestigd;
- het aantal inwoners van de gemeenten waar scholen zijn;
- wie de leerlingen zijn van een school;
- naam, adres en geboortedatum van de leerlingen;
- in welke klas van welk schooltype (b.v. 3e klas havo) een leerling zit;
- welke leraren voor hoeveel uren lesgeven op een school;
- naam, adres, woonplaats, bevoegdheid en salaris van een leraar.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

2. **Rijksbelastingen**

De dienst Rijksbelastingen ontwerpt een database ten behoeve van de inning van opgelegde aanslagen in de inkomstenbelasting. De persoons- en adresgegevens van belastingplichtige inwoners worden in de database opgenomen. De inspecties leggen aanslagen op aan inwoners. Een inwoner ontvangt een gedateerd aanslagbiljet waarop naast aanslaggegevens zoals belastbaar inkomen, het jaar, het te betalen bedrag ook inspectiegegevens zoals spreekuur, telefoon worden vermeld. Een aanslag mag in termijnen worden betaald. De ter-

mijnregeling wordt echter niet in de database opgenomen. Bijgehouden wordt wanneer en hoeveel is betaald op een aanslag. Indien geen of onvoldoende betaling heeft plaatsgevonden bij het verstrijken van een termijn wordt een gedateerde aanmaning gestuurd, waarop het nog te betalen bedrag en de opgelegde boete worden vermeld.

Ontwerp voor deze database een netwerkmodel. Geef van elk recordtype de attributen.

3. **Rederij**

Een rederij zet een database op voor de planning van de door haar schepen te varen routes en de te vervoeren vrachten.
Een route is een rechtstreekse verbinding tussen een vertrek en een aankomsthaven en heeft een bepaalde lengte. Op elke route is er kans op storm, die echter niet op alle routes even groot is. De rederij heeft schepen van verschillende soorten, ieder met hun eigen tonnage, gemiddelde snelheid en benodigd aantal bemanningsleden.
De rederij heeft zich gespecialiseerd in het vervoer van een aantal materialen waarvan bepaalde karakteristieken zoals soortelijke massa bekend zijn. Een schip kan sommige materialen wel en andere niet vervoeren. Er wordt een planning gemaakt van de routes die de schepen zullen gaan bevaren. De planning bevat de vermoedelijke vertrek- en aankomstdatum en de hoeveelheid van de materialen die zullen worden vervoerd.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

4. **Voorraadbeheer**

Een fabriek ontwerpt een database ten behoeve van het voorraadbeheer van de artikelen die door haar afdelingen worden geproduceerd.
Van ieder artikel wordt de voorraad bijgehouden, de nog niet geleverde bestelde hoeveelheden en bijbehorende leverdata.
Voor het maken van artikelen beschikt een afdeling over een aantal machines. Iedere machine is inzetbaar voor alle artikelen die een afdeling kan maken.

De productietijd is niet voor alle artikelen gelijk.
Een artikel moet worden bijgemaakt indien de voorraad minus de bestelde hoeveelheden kleiner is dan de gewenste minimumvoorraad. Echter, omdat een artikel samengesteld kan zijn uit andere artikelen, kan het gebeuren dat eerst één of meer andere artikelen moeten worden gemaakt. Uiteraard kunnen ook deze artikelen weer zijn samengesteld. Vastgesteld wordt het aantal machines dat een afdeling vanaf een zekere datum voor een bepaald aantal dagen zal gaan inzetten voor het produceren van een bepaalde hoeveelheid van een artikel.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

5. **Organisatiestructuur**

Een bedrijf legt in een database gegevens vast over de structuur van de organisatie, de werknemers en de projecten die worden uitgevoerd.

De afdelingen zijn verdeeld in kleinere afdelingen die op hun beurt ook weer onderverdeeld kunnen zijn. Iedere afdeling heeft een budget. Van een medewerker wordt naam, adres en woonplaats vastgelegd. Iedere medewerker behoort tot één afdeling. De chef van de afdeling is tevens de hierarchische chef van de werknemer. Het bedrijf heeft projecten onder handen waaraan medewerkers voor een gedeelte van hun tijd toegevoegd zijn. Van een project wordt vastgelegd de naam, de opdrachtgever en de einddatum. Een van de medewerkers is de projectleider van het project. Vastgelegd wordt van welke projecten een medewerker projectleider is. Niet iedere medewerker van een afdeling kan worden ingezet voor projectwerk. Van de medewerkers die wel kunnen worden ingezet wordt vastgelegd het maximale aantal uren dat ze aan projectwerk mogen besteden. Iedere medewerker heeft een functie. Bij iedere functie hoort een salarisschaal.

Ontwerp een netwerkmodel voor deze database. Geef van elke recordtype de attributen.

6. Bibliotheek

Een bibliotheek ontwerpt een database ten behoeve van de uitleenadministratie en de systeemcatalogus.
De systeemcatalogus is een opsomming van onderwerpen. Ieder onderwerp heeft een systeemnummer, een naam en een omschrijving. Een titel is door één schrijver geschreven en wordt door één uitgever gedrukt en uitgegeven. De filialen (uitleencentra) kunnen van een titel één of meer boeken aanschaffen. Zo'n titel wordt dan in de systeemcatalogus opgenomen onder tenminste één onderwerp. Per filiaal krijgt ieder aangeschaft boek een uniek boeknummer. Voor de aanschaf beschikken de filialen over een eigen budget. Bijgehouden wordt hoeveel geld tot nu toe is besteed.

Lezers krijgen een lezerspas waarop lezersnummer, naam, adres en soort lidmaatschap staan vermeld. Van de uitgeleende boeken worden lezer en datum genoteerd. Indien een lezer een boek niet terugbrengt krijgt deze een boete ter grootte van de aanschafprijs.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

7. Steenbakkerij

Een producent van bakstenen beschikt over een aantal steenfabrieken en over een aantal kleiwinningsgebieden.
Het bedrijf produceert verschillende soorten bakstenen, die door hun naam van elkaar worden onderscheiden. Daar de bakstenen verschillende afmetingen hebben, is de hoeveelheid klei die nodig is om een baksteen te maken niet voor elke soort gelijk. Ook de productietijd is verschillend.

De capaciteit van een steenfabriek - het maximale aantal te produceren stenen per dag - is per fabriek verschillend en afhankelijk van de steensoort. Sommige steensoorten kunnen door bepaalde fabrieken helemaal niet worden geproduceerd.

Er wordt een productieschema gemaakt. Dit houdt in dat voor elke fabriek wordt vastgesteld hoeveel stenen van bepaalde soorten op bepaalde dagen

zullen worden geproduceerd, waarbij uiteraard rekening wordt gehouden met de mogelijkheden van de fabriek. Op deze wijze is de totale hoeveelheid klei die een fabriek op een bepaalde dag nodig heeft bekend. De klei wordt betrokken uit kleiputten, die door hun naam van elkaar te onderscheiden zijn. Uit iedere put kan per dag een zekere hoeveelheid klei worden gewonnen. Om de productie volgens het schema te kunnen uitvoeren wordt ook een verzendschema gemaakt. Dit betekent dat wordt vastgesteld op welke dagen aan welke fabrieken klei wordt geleverd en in welke hoeveelheid.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

8. Uitzendbureau

Een uitzendbureau laat een database ontwerpen ten behoeve van de registratie van vakatures en de vervulling ervan.
Het uitzendbureau levert werknemers voor funkties in bedrijven. Zij heeft daartoe de funkties geclassificeerd en voorzien van een standaardnaam. Een funktie behoort tot een vakgebied. Van de bedrijven waarmee wordt samengewerkt zijn naam, adres, telefoonnummer en contactpersoon bekend.

Een bedrijf meldt vakatures aan. Een vakature is de vraag naar een of meer werknemers voor een funktie, waarbij aangegeven wordt hoeveel uren per week er kan worden gewerkt gedurende welke periode.
Werknemers die zich bij het uitzendbureau melden voor een functie, krijgen een inschrijvingsnummer. Verder worden naam, adres, woonplaats en telefoonnummer genoteerd. Op grond van opleiding en ervaring wordt bepaald voor welke funktie(s) een werknemer in aanmerking komt. De werknemer geeft het uitzendbureau op hoeveel uren per week hij wenst te werken.

Als een ingeschrevene wordt aangesteld op een vakature bij een bedrijf, dan ontvangt hij het bij de funktie behorende uurloon. De aanstelling geschiedt voor een aantal uren per week en kan een andere funktie betreffen dan waarvoor hij staat ingeschreven. Daar ook met deeltijdbanen wordt gewerkt, kan een werknemer bij meer dan één bedrijf werken. Het is bovendien mogelijk dat hij - voor weliswaar een kleiner aantal uren per week - blijft ingeschreven voor een of meer funkties. Een vakature van een bedrijf verdwijnt op het

moment dat met het aantal uren volledig in de vakature is voorzien.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

9. **Transportbedrijf**

Een transportbedrijf gebruikt een database voor de registratie van de in het centrale magazijn aanwezige zendingen en de planning van het vervoer daarvan naar de distriktmagazijnen.
Een zending is bestemd voor één distriktmagazijn, heeft een zeker gewicht, wordt aangeduid met een uniek nummer en wordt in zijn geheel vervoerd.
Voor het vervoer beschikt het transportbedrijf over een aantal vrachtauto's die van elkaar worden onderscheiden door het kenteken.
Deze vrachtauto's maken elke werkdag één rit waarbij een aantal zendingen wordt vervoerd. Uit de te vervoeren zendingen blijkt dan ook welke distriktmagazijnen de vrachtauto's tijdens de rit zullen aandoen.
Om een zo goed mogelijke indeling van de zendingen over de ritten te maken zijn de volgende gegevens nodig:
- de laadvermogens van de vrachtauto's;
- de afstanden tussen het centrale magazijn en de distriktmagazijnen;
- de afstanden tussen de distriktmagazijnen;
- zendingen, al dan niet ingedeeld, de bestemming ervan en de uiterste afleverdatum.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

10. **Productiebedrijf**

Een ondernemening gebruikt ten behoeve van de productie van verschillende typen apparaten een database.
De typen apparaten worden van elkaar onderscheiden door een typenummer en ook door een typenaam met een modelnaam. Van elk type apparaat wordt er keer op keer in een serie een aantal vervaardigd. Alle exemplaren van een serie worden voorzien van eenzelfde serienummer en productiedatum.

Om een serie te kunnen fabriceren is het nodig te weten uit welke onderdelen en in welke aantallen een type apparaat zou moeten bestaan. Indien bij de fabricage van een serie een onderdeel niet voorradig is, kunnen soms andere onderdelen worden gebruikt. Welke onderdelen als vervanging van een onderdeel in een type apparaat dienst kunnen doen, is vastgelegd.
Uiteindelijk bevatten alle exemplaren van een serie dezelfde onderdelen in dezelfde aantallen.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

11. **Spoorwegmaatschappij**

Een spoorwegmaatschappij laat een database ontwerpen ten behoeve van de treindiensten die zij uitvoert op het lijnennet. De stations in het lijnennet zijn voorzien van een unieke stationsnaam. Ieder station beschikt over een aantal perrons en is van een aantal lijnen beginstation, eindstation of tussenstation. De lijnen in het net worden aangeduid met een lijnkode. Een lijn is een verbinding tussen één begin- en één eindstation via één of meer tussenstations en heeft een totale lengte. Op iedere lijn worden diensten onderhouden. Van een dienst wordt vastgelegd:
- het dienstnummer;
- de soort (stoptrein, intercity);
- van het beginstation: de vertrektijd;
- van alle tussenstations waar wordt gestopt: de aankomst- en vertrektijd;
- van het eindstation: de aankomsttijd.

Ontwerp voor deze database een netwerkmodel. Geef van elk recordtype de attributen.

12. **Ziekenhuis**

Een ziekenhuis gebruikt voor de administratie van poliklinische behandelingen van patienten een database.
Het ziekenhuis registreert van patienten NAW-gegevens, geboortedatum en geslacht. Een patient komt onder behandeling van één of meer specialisten.

Iedere specialist heeft zijn eigen specialisme. Gedurende de periode van behandeling kan de specialist verschillende onderzoeken voorschrijven. De verschillende mogelijke onderzoeken (b.v. ECG, darmfoto, bloedonderzoek, etc.) zijn gecodeerd en voorzien van een naam. Een patient krijgt voordat een onderzoek wordt uitgevoerd een beschrijving ervan en krijgt te horen wat de vermoedelijke behandelingsduur zal zijn.
Tijdens een behandeling kunnen medicijnen worden voorgeschreven. Voor ieder medicijn afzonderlijk wordt een recept verstrekt waarop de dosis is vermeld. De data van de tijdens de behandeling verrichte onderzoeken en verstrekte recepten moeten kunnen worden opgevraagd.
Het is mogelijk dat een specialist voor een behandeling één collega inschakelt als vervanger.

Ontwerp voor deze datbase een netwerkmodel. Geef van elk recordtype de attributen.

13. **Luchtbebakening**

De bebakening van luchtwegen ten behoeve van de luchtvaartnavigatie wordt in een database vastgelegd.
Bakens zijn zenders die een radiografisch signaal van een bepaalde frequentie uitzenden. Een baken wordt geïdentificeerd met een bakenkode. Van de bakens wordt vastgelegd van welke luchtwegen ze beginbaken, eindbaken en/of verbindingsbaken zijn. Een luchtweg is de verbinding tussen twee - elk op een vliegveld geplaatste - bakens via één of meer tussenliggende bakens. Aan de luchtwegen zijn unieke namen gegeven.

Een vliegtuig dat een luchtweg gaat volgen moet over de volgende gegevens kunnen beschikken:

- de lengte van de luchtweg;
- de bakens waar het zich telkens op moet richten;
- de afstand tussen twee opeenvolgende bakens;
- per baken de frequentie van het signaal en de coördinaten.

Als een baken van een luchtweg buiten werking is, dan wordt de positie bepaald met behulp van één of meer vervangende bakens.

Ontwerp een netwerkmodel voor deze database. Geef van elk recordtype de attributen.

9. CODASYL-DDL

9.1. Inleiding

Het CODASYL COBOL Committee (CC) is verantwoordeijk voor de ontwikkeling van COBOL. Een werkgroep van het CC, de Data Base Task Group (DBTG), publiceerde in 1971 een rapport met voorstellen voor drie verschillende talen: een schema data description language (schema DDL), een subschema data description language (subschema DDL) en een data manipulation language (DML).

De schema DDL is een taal waarmee de structuur van een netwerk-database kan worden beschreven. Zo'n beschrijving, in de DBTG-terminologie een schema genoemd, komt dus overeen met het eerder besproken begrip conceptual schema.

Een subschema DDL is een taal waarmee dat deel van de database kan worden beschreven dat voor een bepaalde applicatie van belang is. Het subschema komt derhalve overeen met het begrip external schema. De voorgestelde subschema DDL is bedoeld om te gebruiken in samenhang met COBOL. Het voorstel moet dan ook worden gezien als een uitbreiding van die taal.

Ook de DML is voorgesteld in een vorm die geschikt is om de opdrachten in te bedden in een COBOL programma. De DML is derhalve geen zelfstandige taal, maar bevat uitsluitend opdrachten voor database manipulatie.
De DBTG sprak de hoop uit dat de voor andere programmeertalen verantwoordelijke organisaties voorstellen zouden doen voor een bij die talen passende subschema DDL en DML. Tot nu toe is daar niet veel van terechtgekomen.

Het DBTG-rapport is een belangrijke stimulans geweest voor de ontwikkeling van op netwerken gebaseerde database systemen.
Sinds 1971 bestaat er een nieuwe CODASYL commissie, het Data Description Language Committee (DDLC), dat verantwoordelijk is voor de ontwikkeling van de DDL, op dezelfde wijze als CC dat is voor COBOL.
De COBOL DML en subschema DDL zijn nu opgenomen in het COBOL Journal of Development onder de naam The Cobol Data Base Facility.

In de volgende paragrafen wordt een aantal aspecten behandeld van de DDL uit het rapport van de CODASYL DDLC (Journal of Development 1981).

9.2. Achtergronden

In het netwerkmodel wordt vastgelegd welke de voor de toepassingen noodzakelijke recordtypen zijn en welke samenhang er tussen de recordtypen bestaat. Om met de bijbehorende database te kunnen manipuleren zijn record-en setbeschrijvingen nodig. In een bestandsomgeving worden recordbeschrijvingen alleen teruggevonden in de programma's. In een database omgeving zal met een programma meestal niet met records van alle recordtypen worden gemanipuleerd, doch slechts met records van enkele recordtypen. Alleen de recordbeschrijvingen van deze recordtypen zullen in het programma moeten worden opgenomen. Uiteraard blijft een record- en setbeschrijving van de gehele database nodig. Ten behoeve van de programma's worden een of meer subschema's gemaakt die een beschrijving van een deel van de database bevatten. In de programma's wordt een kopie van de recordbeschrijving uit het bijbehorende subschema opgenomen in de z.g. user working area (UWA). Het transport van gegevens tussen een programma en de database vindt altijd plaats via de UWA. In het programma worden daartoe z.g. data manipulation language (DML) opdrachten opgenomen (zie hoofdstuk 10). Naast DML opdrachten bevat het programma ook nog COBOL opdrachten om met de verkregen gegevens te kunnen manipuleren.

9.3. Schema

Een schema bevat in de aangegeven volgorde:
- één schema entry;
- één of meer record entries;
- nul, één of meer set entries.

In de schema entry wordt het schema beschreven, in een record entry een recordtype en in een set entry een settype. Er zijn dus net zoveel record entries als er recordtypen zijn en net zo veel set entries als settypen.

Bij de hierna volgende beschrijving van de entries is een groot aantal details uit het DBTG-rapport niet vermeld. Alleen de essentie is weergegeven.

9.4. Schema entry

De schema entry bevat een clausule waarmee het schema een naam wordt gegeven:

SCHEMA NAME IS schemanaam

9.5. Record entry

Een record entry bevat in de aangegeven volgorde:
- één record subentry;
- nul, één of meer data subentries.

9.5.1. Record subentry

Een record subentry bevat in de aangegeven volgorde:
- één record clausule waarmee aan het recordtype een naam wordt gegeven:
  RECORD NAME IS recordnaam
- nul, één of meer key clausules:
  [KEY keynaam IS {datanaam}... DUPLICATES ARE [NOT] ALLOWED] ...

Met KEY wordt aangegeven dat het recordtype een sleutel met naam keynaam heeft en dat deze sleutel bestaat uit de combinatie van de genoemde data-items die uiteraard deel moeten uitmaken van het desbetreffende recordtype.
Met DUPLICATES wordt aangegeven of het al dan niet wordt toegestaan om recordoccurrences met gelijke waarde voor de sleutel keynaam in de database op te slaan.

N.B. In een CODASYL omgeving wordt met een sleutel meer een index bedoeld dan een werkelijke sleutel, zoals in het relationele model.

9.5.2. Data subentry

Een data subentry bevat de clausule:

datanaam TYPE IS { DECIMAL / CHARACTER } integer

Hiermee wordt aangegeven dat het betreffende data-item deel uitmaakt van het recordtype en dat het data-item uit het opgegeven aantal cijfers of karakters bestaat.

9.6. Set entry

Een set entry bevat in de aangegeven volgorde:
- één set subentry;
- één member subentry.

In de set subentry wordt een algemene beschrijving van het settype gegeven en in de member subentry een beschrijving van de member van het settype.

9.6.1. Set subentry

De set subentry bevat de volgende clausules in de aangegeven volgorde:
- één set clausule waarmee aan het settype een naam wordt gegeven:
  SET NAME IS setnaam
- één owner clausule waarmee wordt aangegeven welk recordtype de owner is van het settype:
  OWNER IS recordnaam
- één order clausule:

$$\text{ORDER FOR INSERTION IS} \left\{ \begin{array}{l} \text{FIRST} \\ \text{LAST} \\ \text{SORTED} \end{array} \right\}$$

Hiermee wordt de volgorde van de memberoccurrences in iedere setoccurrence van het settype aangegeven. De volgorde wordt gerealiseerd door in de recordoccurrences verwijzingen (eng.: pointers) naar het volgend recordoccurrence op te nemen. Het DBMS zorgt er voor dat bij het opvoeren, verwijderen en wijzigen van recordoccurrences de volgorde gehandhaafd blijft. De volgorde van de memberoccurrences hangt af van de opgegeven specificatie.

FIRST : in omgekeerde volgorde van opvoering. Een nieuw memberoccurrence komt als eerste in de setoccurrence.

LAST : in volgorde van opvoering. Een nieuw memberoccurrence komt als laatste in de setoccurrence.

SORTED : in volgorde van de waarde van de sorteersleutel. Een nieuw memberoccurrence komt ná de memberoccurrences met een kleinere waarde van de sorteersleutel en vóór de memberoccurrences met een grotere waarde van de sorteersleutel. De sorteersleutel zelf wordt gespecificeerd in de member subentry van deze set entry.

9.6.2. Member subentry

Een member subentry bevat de volgende clausules in de aangegeven volgorde:

- één member clausule waarmee wordt aangegeven welk recordtype de member is van het settype:

  MEMBER IS recordnaam

- één insertion clausule

  INSERTION IS { AUTOMATIC / MANUAL }

Hiermee wordt aangegeven of een memberoccurrence bij opvoering in de database al dan niet wordt opgenomen in een setoccurrence.

AUTOMATIC: de memberoccurrence wordt opgenomen in de setoccurrence die wordt bepaald op grond van de SET SELECTION clausule (zie het einde van deze paragraaf)

MANUAL : de memberoccurrence wordt niet opgenomen in een setoccurrence (zie hiervoor sometime membership).

- één retention clausule

  RETENTION IS { FIXED / MANDATORY / OPTIONAL }

Hiermee wordt aangegeven of een memberoccurrence die eenmaal is opgenomen in een setoccurrence, al dan niet deel uit moet blijven maken van een setoccurrence.

FIXED : een memberoccurrence blijft altijd deel uitmaken van dezelfde setoccurrence.

MANDATORY: een memberoccurrence blijft altijd deel uitmaken van een setoccurrence. Een verhuizing van de ene naar de andere setoccurrence is echter wel mogelijk.

OPTIONAL : een memberoccurrence hoeft geen deel uit te maken van een setoccurrence. Het is mogelijk om een memberoccurrence uit de setoccurrence te halen en ook om een memberoccurrence in een setoccurrence te brengen.

- nul, één of meer duplicate clausules

  [DUPLICATES ARE NOT ALLOWED FOR {datanaam}...]...

Hiermee wordt aangegeven dat binnen een setoccurrence geen memberoccurrences mogen voorkomen met dezelfde waarde van de opgegeven combinatie van data-items.

- nul of één key clausule:

  KEY IS { ASCENDING / DESCENDING } datanaaml [[ASCENDING / DESCENDING] datanaam2] ...

  [DUPLICATES ARE { FIRST / LAST / NOT ALLOWED }]

Deze clausule moet aanwezig zijn als er bij de orderclausule in de set subentry ORDER IS SORTED is opgegeven. De clausule dient om de sorteersleutel te definiëren en om aan te geven wat er gedaan moet worden als een record aan de database wordt toegevoegd met een waarde voor de sorteersleutel die al in de setoccurrence voorkomt.
Deze key clausule moet niet worden verward met die uit de record subentry.

- één selection clausule:

  SET SELECTION OWNER IDENTIFIED BY
  { APPLICATION / KEY keynaam }

Hiermee wordt aangegeven op welke manier het DBMS de setoccurrences bepaalt bij

- het opvoeren van een memberoccurrence met insertion automatic in de database (zie STORE paragraaf 10.9.);
- het inbrengen van een memberoccurrence in een setoccurrence (zie CONNECT paragraaf 10.6.);
- het zoeken van een bepaalde memberoccurrence in een setoccurrence (zie FIND WITHIN USING paragraaf 10.8.4.).

De setoccurrence die wordt bepaald, is bij

- APPLICATION: de setoccurrence waarin de laatste benaderde recordoccurrence van het owner of membertype is opgenomen;

- KEY : de setoccurrence waarvan de owneroccurrence een waarde voor de opgegeven key heeft gelijk aan de waarde van de opgegeven key in de UWA (zie paragraaf 10.2.). De betreffende key moet in de record subentry (zie paragraaf 9.4.1.) uiteraard met DUPLICATES NOT ALLOWED zijn gespecificeerd.

10. CODASYL-DML

10.1. Inleiding

Voor het manipuleren met een database die gebaseerd is op de CODASYL voorstellen, is een groot aantal DML-opdrachten aanwezig. Een aantal hiervan zal in dit hoofdstuk alleen maar worden vermeld. Uitsluitend de opdrachten FIND, GET, STORE, MODIFY en ERASE zullen worden behandeld, zij het dat verschillende opties hierbij worden weggelaten. Ook de aspecten die te maken hebben met het gelijktijdig gebruiken van de database door verschillende gebruikers blijven buiten beschouwing. Verder is er in de voorbeelden voor de eenvoud van uitgegaan dat schema en subschema identiek zijn.

10.2 Run-unit

Bij het compileren wordt een programma met DML-opdrachten automatisch uitgebreid met een UWA. Dit geheel (programma plus UWA) heet een run-unit. De UWA is een afspiegeling van het bijbehorend subschema en bevat o.a.:

- voor ieder recordtype in het subschema een recordbeschrijving waarmee o.a. wordt aangegeven uit welke data items een record bestaat en het formaat ervan;
- één error status indicator: een register waarin het DBMS na afloop van iedere DML opdracht vastlegt op welke manier de DML opdracht is beëindigd (b.v. record gevonden, foutmelding);
- een aantal currency indicatoren: registers waarin het DBMS na afloop van sommige DML opdrachten vastlegt met welk record in de database is gemanipuleerd (niet de inhoud van het record!);

Transport van gegevens tussen de database en de run-unit vindt altijd plaats via de UWA. Indien een recordoccurrence uit de database wordt opgevraagd (DML-opdracht) wordt de inhoud ervan door het DBMS in de UWA op de plaats van het betreffende recordtype gezet. Manipulatie met een recordoccurrence in de UWA geschiedt met z.g. host language opdrachten (b.v. COBOL). Indien een recordoccurrence aan de database wordt toegevoegd (DML-opdracht) wordt de inhoud van het betreffende record in de UWA door het DBMS naar de database gebracht. Met behulp van host language opdrachten moet dus eerst aan het record in de UWA een inhoud zijn gegeven.

10.3. Database key

Aan ieder record dat in de database wordt opgeslagen, wordt door het DBMS een uniek nummer toegekend. Dit nummer heet de database key van het record. Gedurende de periode dat een recordoccurrence in de database aanwezig is, behoudt het dezelfde database key m.a.w. de database key van een record kan niet worden gewijzigd.
De database key is voor het DBMS een belangrijk hulpmiddel bij het selecteren van records, omdat op basis van de database key de fysieke plaats van het record in de database kan worden bepaald. Voor de gebruiker (programma) speelt de database key een ondergeschikte rol. Hij hoeft geen weet te hebben van de representatie van een database key van een record om er mee te kunnen manipuleren, maar moet wel op de hoogte zijn van het bestaan ervan.

10.4. Currency indicatoren

Bij manipulatie met records in een bestand is het altijd duidelijk tot welk recordtype een record behoort. In een database zijn daarentegen records van verschillende recordtypen opgeslagen en kan een record deel uitmaken van setoccurrences van verschillende settypen. Daar bij opeenvolgende DML-opdrachten records van verschillend type kunnen zijn betrokken die bovendien deel uit kunnen maken van verschillende setoccurrences, worden door het DBMS zgn. current records bijgehouden. Hiertoe worden in de UWA van een run-unit de volgende currency indicatoren opgenomen:
- één currency indicator van de run-unit;
- één currency indicator voor ieder recordtype;
- één currency indicator voor ieder settype.

Een currency indicator is een register waarin de database key van een record kan staan.
Vóór de uitvoering van de eerste DML-opdracht in een run-unit hebben alle currency indicatoren een ongedefinieerde waarde, de zgn. nulwaarde.

Sommige currency indicatoren worden door het DBMS bijgewerkt na afloop van de succesvolle uitvoering van bepaalde DML-opdrachten. Welke currency indicatoren worden bijgewerkt hangt af van de betreffende DML-opdracht en van het betreffende record waarmee is gemanipuleerd. Het bijwerken kan echter selectief worden onderdrukt. De currency indicatoren worden niet bijgewerkt

indien de DML-opdracht niet met succes kan worden uitgevoerd.
Aldus bevat een currency indicator of de nulwaarde of de database key van een record en wel de currency indicator van:

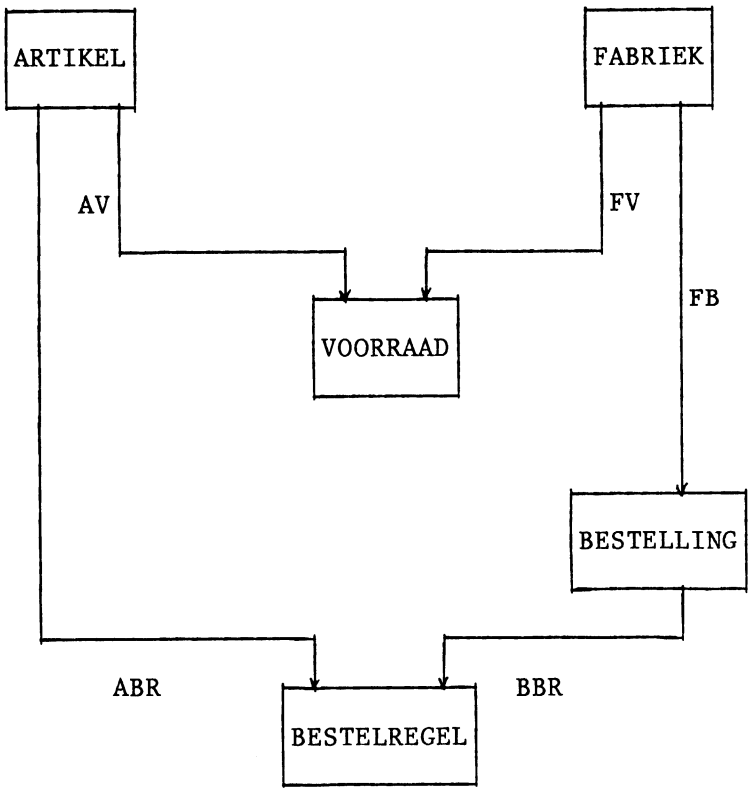
- de run-unit : het laatste door de run-unit benaderde record, ongeacht het type van het record; dit record heet het current record van de run-unit (CRRU).
- een recordtype : het laatste door de run-unit benaderde record van het betreffende recordtype; dit record heet het current record van het betreffende recordtype (CRRT recordnaam).
- een settype : het laatste door de run-unit benaderde record van het owner of membertype van het betreffende settype, onder de voorwaarde dat het record deel uitmaakt van een set-occurrence van het betreffende settype; dit record heet het current record van het betreffende settype (CRST setnaam).

Bij het manipuleren spelen de currency indicatoren een zeer belangrijke rol. De afloop van de meeste DML-opdrachten is afhankelijk van de inhoud van currency indicatoren.

10.5. Voorbeeld-database

De database met behulp waarvan de verschillende DML-opdrachten zullen worden toegelicht, komt in grote lijnen overeen met de database die in par. 6.2. is beschreven.

Het netwerkmodel van deze database is:



Het schema luidt:

```
SCHEMA VOORBEELD-DATABASE

RECORD ARTIKEL
    KEY ARTNR IS ART# DUPLICATES NOT ALLOWED
    ART#
    ARTNAAM
    CATEGORIE

RECORD FABRIEK
    KEY FABNR IS FAB# DUPLICATES NOT ALLOWED
    FAB#
    FABNAAM
    PLAATS

RECORD VOORRAAD
    VRD
    MINVRD
```

```
RECORD BESTELLING
     KEY BESTNR IS B# DUPLICATES NOT ALLOWED
     B#
     KLANT
     DATUM

RECORD BESTELREGEL
     HOEV
     PRIJS

SET AV
     OWNER ARTIKEL
     ORDER FIRST
     MEMBER VOORRAAD INSERTION AUTOMATIC RETENTION FIXED
     SET SELECTION BY APPLICATION

SET FV
     OWNER FABRIEK
     ORDER FIRST
     MEMBER VOORRAAD INSERTION AUTOMATIC RETENTION FIXED
     SET SELECTION BY KEY FABNR

SET ABR
     OWNER ARTIKEL
     ORDER FIRST
     MEMBER BESTELREGEL INSERTION AUTOMATIC RETENTION FIXED
     SET SELECTION BY KEY ARTNR

SET FB
     OWNER FABRIEK
     ORDER LAST
     MEMBER BESTELLING INSERTION AUTOMATIC RETENTION FIXED
     DUPLICATES ARE NOT ALLOWED FOR KLANT,DATUM
     SET SELECTION BY APPLICATION

SET BBR
     OWNER BESTELLING
     ORDER FIRST
     MEMBER BESTELREGEL INSERTION AUTOMATIC RETENTION FIXED
     SET SELECTION BY APPLICATION
```

Deze database bevat de volgende recordoccurrences.

ARTIKEL

art#	artnaam	categorie
A1	puzzel	speelgoed
A2	schop	tuin
A3	hijskraan	speelgoed
A4	flesopener	huishouding
A5	verfboek	speelgoed

FABRIEK

fab#	fabnaam	plaats
F1	Tifa	Rotterdam
F2	Lepo	Breda
F3	Drof	Tilburg
F4	Botlat	Utrecht
F5	Trocine	Maastricht

VOORRAAD

fab#	art#	vrd	minvrd
F1	A1	200	150
F1	A2	300	75
F1	A4	150	300
F2	A1	700	200
F2	A2	875	100
F2	A3	0	20
F2	A4	210	210
F2	A5	370	180
F3	A3	15	30
F3	A4	175	100
F4	A4	300	100

BESTELLING

b#	klant	fab#	datum
B1	Smit	F4	16-04-83
B2	Slager	F3	25-08-83
B3	Snoek	F2	17-10-83
B4	Staal	F1	04-05-83
B5	Slager	F2	19-11-83
B6	Smit	F2	17-10-83

BESTELREGEL

*b#	*art#	hoev	prijs
B1	A4	200	4.50
B2	A3	150	3.50
B2	A4	75	4.75
B3	A1	200	2.75
B3	A2	100	2.--
B3	A3	100	3.50
B3	A4	50	5.--
B3	A5	1000	0.50
B4	A2	300	2.--
B4	A4	100	5.--
B5	A1	200	2.75
B5	A2	300	1.75
B5	A5	400	0.50
B6	A1	400	2.50
B6	A3	200	3.25

Voor de duidelijkheid zijn de occurrences van het recordtype VOORRAAD aangevuld met de bijbehorende waarden van fab# en art#, de occurrences van het recordtype BESTELLING met de bijbehorende waarden van fab#, en de occurrences van het recordtype BESTELREGEL met de waarden van b# en art#, alhoewel in de database deze attribuutwaarden niet zijn opgenomen. Bij welke fabriek een recordoccurrence van het recordtype VOORRAAD hoort blijkt uit de setoccurrence van FV waarin het is opgenomen. De setoccurrences worden niet hier getekend, doch indien nodig bij de voorbeelden.

In de voorbeelden gaat het er uiteraard om de toepassing van DML-opdrachten te illustreren. De juiste vorm van de in de gasttaal geschreven opdrachten - de taal waarin de DML-opdrachten zijn ingebed - is hierbij minder belangrijk. In de voorbeelden worden deze opdrachten in COBOL geschreven.
Er wordt in de voorbeelden van uitgegaan dat de met name genoemde records in de database aanwezig zijn, om te voorkomen dat allerlei testopdrachten moeten worden toegevoegd.

10.6. Overzicht van enige DML-opdrachten

CONNECT	:	neemt een reeds in de database aanwezig record op als member in één of meer setoccurrences.
DISCONNECT	:	verwijdert een record als member uit één of meer setoccurrences; het record blijft aanwezig in de database.
ERASE	:	verwijdert één of meer records uit de database.
FETCH	:	selecteert één record uit de database en brengt het over naar de UWA (een combinatie van de FIND en de GET-opdracht).
FIND	:	selecteert één record uit de database.
GET	:	brengt een geselecteerd record naar de UWA.
MODIFY	:	wijzigt de inhoud van een record in de database.
RECONNECT	:	plaatst een memberrecord over naar een andere setoccurrence van het settype.
STORE	:	brengt een record van de UWA naar de database; het record wordt eventueel ook als member in één of meer setoccurrences opgenomen.

10.7. GET-opdracht

Het formaat van deze opdracht is:

GET recordnaam

Functie:
Indien het current record van de run-unit (CRRU) van het type recordnaam is, wordt het CRRU uit de database opgehaald en overgebracht naar de UWA. De error status indicator krijgt dan de waarde 0.

Indien het CRRU niet van het type recordnaam is wordt de GET-opdracht niet uitgevoerd en wordt in de error status indicator de foutcode van de betreffend foutmelding gezet.
Na afloop van de al dan niet succesvolle uitvoering van een GET-opdracht worden de currency indicatoren niet bijgewerkt.

10.8. FIND-opdracht algemeen

De FIND-opdracht wordt gebruikt om een bepaald record in de database te selecteren. Selecteren houdt hier in het bepalen van de database key van het gevraagde record. Indien het gevraagde record niet in de database aanwezig is dan krijgt de error status indicator de waarde EOS (end of set) of NF (not found) afhankelijk van het soort FIND-opdracht. Indien het gevraagde record wel in de database aanwezig is krijgt de error status indicator de waarde 0 (≡ succesvolle uitvoering van een DML-opdracht).

Het record wordt niet naar de UWA gebracht. Wil men over de inhoud van het record kunnen beschikken, dan moet nog een GET-opdracht worden gegeven.

De algemene gedaante van de FIND-opdracht is:

```
FIND record-selection-expression
[RETAINING    [{ |RECORD          | } ] CURRENCY]
              [{ |{set-name}...   | } ]
```

De record-selection-expression wordt gebruikt om aan het DBMS de criteria bekend te maken op grond waarvan het gevraagde record kan worden ge

selecteerd. Er bestaan verschillende vormen (formats) van deze expressie. Deze zullen in de volgende sub-paragrafen worden besproken.

Wordt nu m.b.v. een FIND-opdracht een record geselecteerd dan gebeurt het volgende:

- de database key van het geselecteerde record wordt gezet in de currency indicator van de run-unit (current record van de run-unit);
- de database key van het geselecteerde record wordt gezet in de currency indicator van het bijbehorende recordtype (current record van recordtype);
- de database key van het geselecteerde record wordt geplaatst in de currency indicator(en) van de settype(n) (current record van settype), waarvan het een owner- of memberoccurrence is in een setoccurrence.

De RETAINING-clausule kan worden gebruikt om het bijwerken van een of meer currency indicatoren te onderdrukken. RETAINING RECORD betekent dat het CRRT van het betreffende recordtype niet wordt bijgewerkt, terwijl bij RETAINING setnaam het CRST van setnaam ongewijzigd blijft.

10.8.1. Selectie van een (volgend) record van een bepaald recordtype

Het formaat van deze opdracht is

FIND { ANY / DUPLICATE } recordnaam USING {datanaam} ...

Functie:
Indien ANY is gespecificeerd, wordt het eerste het beste record van het genoemde type geselecteerd waarvan de inhoud van de data-item(s) gelijk is aan een opgegeven waarde. Deze waarde moet worden geplaatst in de overeenkomstige velden in de UWA. Indien meer dan één record in aanmerking komt, kiest het DBMS er een op grond van de door het systeem gehanteerde volgorde. Indien er geen record is met de opgegeven waarde dan krijgt de error status de waarde NF (not found).

Indien DUPLICATE is gespecificeerd, wordt het eerstvolgende record - volgend op het current record van het recordtype - geselecteerd waarvan de inhoud van de data-item(s) gelijk is aan de inhoud van het (de) overeenkomstige data-item(s) in het current record van het recordtype. Indien er geen record meer is dat hieraan voldoet dan krijgt de error status de waarde NF.

Voorbeelden.

1. De gegevens van een artikel uit de categorie speelgoed worden verkregen met:

```
MOVE SPEELGOED TO CATEGORIE
FIND ANY ARTIKEL USING CATEGORIE
GET ARTIKEL
PRINT (ART#, ARTNAAM)
```

Afgedrukt wordt dan
al puzzel
of a3 hijskraan
of a5 verfboek
afhankelijk van de volgorde die het DBMS hanteert.

2. Maak een lijstje van de art#'s en artnamen van de artikelen behorende tot de categorie speelgoed.

```
(0) MOVE SPEELGOED TO CATEGORIE
(1) FIND ANY ARTIKEL USING CATEGORIE
    ZOLANG - NF DOE
    ┌
    │GET ARTIKEL
    │PRINT (ART#,ARTNAAM)
(2) │FIND DUPLICATE ARTIKEL USING CATEGORIE
    └
```

Indien bij de uitvoering van dit programma de volgende afdruk wordt verkregen:

a1 puzzel
a3 hijskraan
a5 verfboek

dan is het verloop van de currency indicatoren als volgt geweest:

	CRRU	CRRT ARTIKEL
(0)	null	null
(1)	a1	a1
(2)	a3	a3
(2)	a5	a5
(2)	a5	a5

ERROR STATUS NF

N.B. In de currency tabel worden hier voor de duidelijkheid de sleutelwaarden van de records genoteerd en niet de database keys!

Indien met FIND ANY ARTIKEL USING CATEGORIE niet a1 maar a3 of a5 zou worden geselecteerd, dan zal het CRRU in een andere volgorde de database keys van a1, a3 en a5 bevatten. In deze volgorde zullen de records dan ook worden afgedrukt.

10.8.2. Selectie van een "volgend" record in een setoccurrence.

Het formaat van deze opdracht is:

FIND { NEXT / PRIOR / FIRST / LAST } recordnaam WITHIN setnaam

Functie:
Er wordt een memberrecord geselecteerd van de setoccurrence die bepaald wordt door het current record van het opgegeven settype. De setoccurrence wordt hier dus niet bepaald door de setselectie clausule in het schema.

Bij FIRST en LAST wordt het eerste resp. het laatste memberrecord van de setoccurrence geselecteerd.
Bij NEXT en PRIOR is dat het record volgend op resp. voorafgaand aan het current record van het settype. Is het current record van de set het owner-record, dan is de werking gelijk aan die van FIRST resp. LAST.

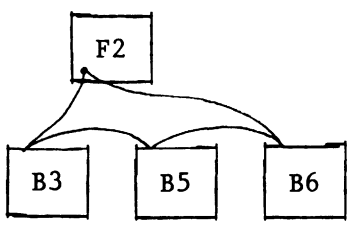
Indien er geen eerste, laatste, volgend resp. voorafgaand record in de set-occurrence (meer) is, krijgt de error status de waarde EOS (end of set).

Voorbeelden.

1. Maak een lijst van de klanten die bij fabriek F2 na 18 oktober 1983 een bestelling hebben geplaatst.

```
(0) MOVE F2 TO FAB#
(1) FIND ANY FABRIEK USING FAB#
(2) FIND FIRST BESTELLING WITHIN FB
    ZOLANG ¬ EOS DOE
    ┌
    │GET BESTELLING
    │IF DATUM > 831018 THEN PRINT (KLANT)
(3) │FIND NEXT BESTELLING WITHIN FB
    └
```

De setoccurrence van het settype FB waarvan fabriek F2 de owner is, ziet er als volgt uit:



Het verloop van de currency indicatoren is daarom:

	CRRU	CRRT FABRIEK	CRRT BESTELLING	CRST FB	
(0)	null	null	null	null	
(1)	F2	F2	null	F2	
(2)	B3	F2	B3	B3	
(3)	B5	F2	B5	B5	
(3)	B6	F2	B6	B6	
(3)	B6	F2	B6	B6	ERROR STATUS: EOS

Afgedrukt wordt dus:

Slager

2. De totale voorraad van artikel A4 wordt verkregen met

```
MOVE 0 TO TOTAAL
MOVE A4 TO ART#
FIND ANY ARTIKEL USING ART#
FIND FIRST VOORRAAD WITHIN AV
ZOLANG ¬ EOS DOE
┌─
│GET VOORRAAD
│ADD VRD TO TOTAAL
│FIND NEXT VOORRAAD WITHIN AV
└─
PRINT (ART#, TOTAAL)
```

Afgedrukt wordt:

A4 835

10.8.3. Selectie van de owner van een setoccurrence

Het formaat van deze opdracht is:

FIND OWNER WITHIN setnaam

Functie:
Het record dat geselecteerd wordt is de owner van de setoccurrence die bepaald wordt door het current record van het settype. Bevat deze setoccurrence geen ownerrecord dan krijgt de error status de waarde NF.

Voorbeelden.

1. De fabriek waar klant Slager op 25-08-1983 een bestelling heeft geplaatst.

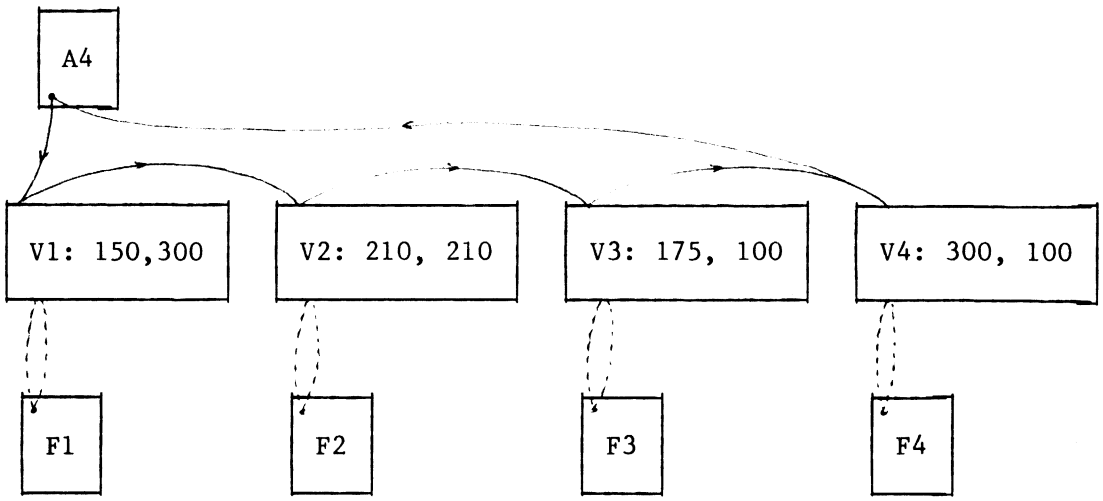
```
MOVE SLAGER TO KLANT
MOVE 830825 TO DATUM
FIND ANY BESTELLING USING KLANT,DATUM
FIND OWNER WITHIN FB
GET FABRIEK
PRINT (FAB#)
```

Afgedrukt wordt:
F3

2. Maak een lijst van de fabrieken waarbij de voorraad van artikel A4 kleiner is dan de gewenste minimum voorraad.

```
    MOVE A4 TO ART#
(1) FIND ANY ARTIKEL USING ART#
(2) FIND FIRST VOORRAAD WITHIN AV
    ZOLANG ¬ EOS DOE
    ┌
    │ GET VOORRAAD
    │ IF VRD < MINVRD THEN
    │ ┌
(3) │ │ FIND OWNER WITHIN FV
    │ │ GET FABRIEK
    │ │ PRINT (FAB#, VRD, MINVRD)
    │ └
(4) │ FIND NEXT VOORRAAD WITHIN AV
    └
```

De setoccurrence van het settype AV waarvan artikel A4 owner is ziet er als volgt uit:



Hierin zijn de (onvolledige) setoccurrences van het settype FV met behulp van stippellijnen aangegeven.

Het verloop van de currency tabel is als volgt:

	CRRU	CRRT ARTIKEL	CRRT VOORRAAD	CRRT FABRIEK	CRST AV	CRST FV	
(1)	A4	A4	--	--	A4	--	
(2)	V1	A4	V1	--	V1	V1	
(3)	F1	A4	V1	F1	V1	F1	
(4)	V2	A4	V2	F1	V2	V2	
(4)	V3	A4	V3	F1	V3	V3	
(4)	V4	A4	V4	F1	V4	V4	
(4)	V4	A4	V4	F1	V4	V4	EOS

Afgedrukt wordt dus:

F1 150 300

10.8.4. Selectie van een bepaald record in een setoccurrence

Het formaat van deze opdracht is:

FIND recordnaam WITHIN setnaam USING {datanaam}...

Functie:

Er wordt een bepaald memberrecord geselecteerd in de setoccurrence, die wordt bepaald met behulp van de setselectie clausule. De grootheid die het DBMS hiervoor nodig heeft moet dan de juiste waarde hebben. Afhankelijk van de opgegeven setselectie clausule is deze grootheid:

by KEY: de sleutelwaarde van de owner van de gewenste setoccurrence.

by APPLICATION: de currency indicator van het opgegeven settype.

Het eerste het beste memberrecord waarvan de inhoud van de data-item(s) gelijk is aan de inhoud van de overeenkomstige velden in de UWA, wordt geselecteerd.

Het zoeken begint bij het eerste memberrecord en verloopt verder volgens de voor het settype gespecificeerde ordening.

Voorbeeld.

1. Het bestelnummer van de bestelling op 25-08-1983 van klant Slager bij fabriek F3.

```
MOVE F3 TO FAB#
FIND ANY FABRIEK USING FAB#
{nodig voor de bepaling van de s.o.
waarin moet worden gezocht}
MOVE SLAGER TO KLANT
MOVE 830825 TO DATUM
FIND BESTELLING WITHIN FB USING KLANT,DATUM
GET BESTELLING
PRINT (B#)
```

Afgedrukt wordt:
B2

10.8.5. Selectie van een record met dezelfde waarde in een setoccurrence

Het formaat van deze opdracht is:

FIND DUPLICATE WITHIN setnaam USING {datanaam}...

Functie:
Het memberrecord van de setoccurrence die bepaald wordt door het current record van het settype, waarvan de inhoud van de data-item(s) gelijk is aan de inhoud van het (de) overeenkomstige data-item(s) van het current record van het settype, wordt geselecteerd. De setoccurrence waarin wordt gezocht, wordt dus niet bepaald door de setselectie clausule.

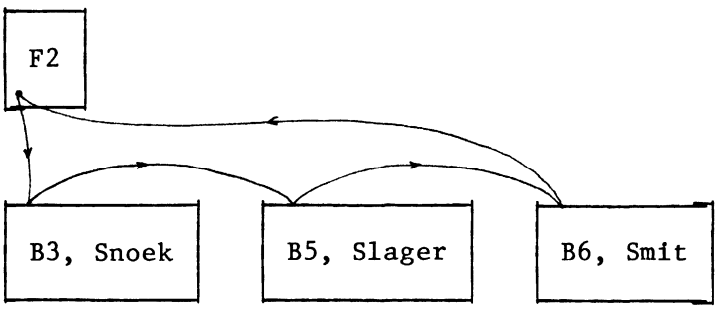
Het zoeken begint vanaf het current record van het settype en verloopt verder volgens de voor het settype gespecificeerde ordening.

Voorbeelden.

1. Maak een lijst van de bestelnummers van alle bestellingen van klant Smit bij fabriek F2.

```
    MOVE F2 TO FAB#
(1) FIND ANY FABRIEK USING F2
    {nodig voor de bepaling van de s.o.
    waarin moet worden gezocht}
    MOVE SMIT TO KLANT
(2) FIND BESTELLING WITHIN FB USING KLANT
    ZOLANG ¬ EOS DOE
    ┌
    │GET BESTELLING
    │PRINT(B#)
(3) │FIND DUPLICATE WITHIN FB USING KLANT
    └
```

De setoccurrence van FB waarvan fabriek F2 de owner is ziet er als volgt uit:



Als gevolg hiervan is het verloop van de currency registers:

	CRRU	CRRT FABRIEK	CRRT BESTELLING	CRST FB	
(1)	F2	F2	--	F2	
(2)	B6	F2	B6	B6	
(3)	B6	F2	B6	B6	EOS

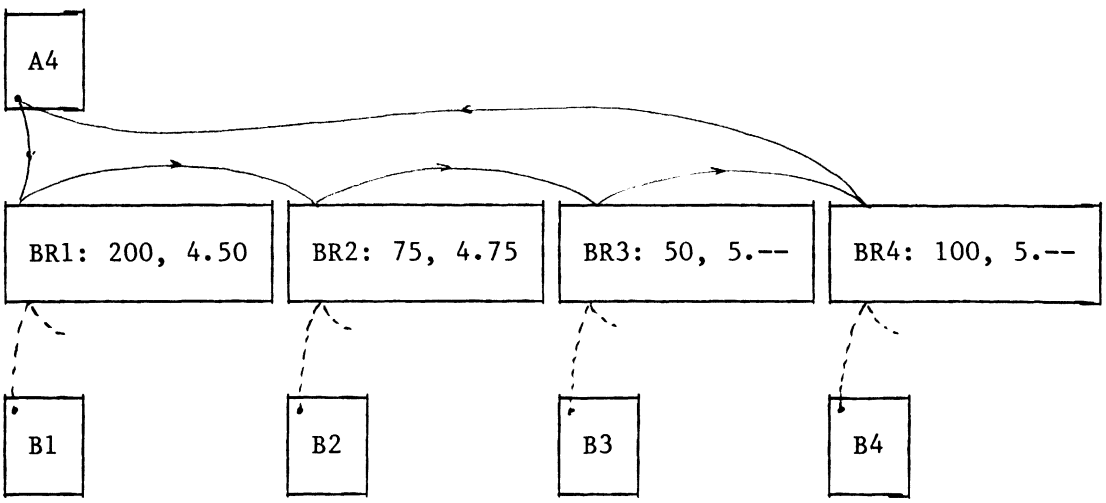
Afgedrukt wordt dus:

B6

2. Maak een lijst van de bestelnummers van de bestellingen waarop artikel A4 tegen een prijs van fl. 5,-- is besteld.

```
    MOVE A4 TO ART#
    {nodig voor de bepaling van de s.o.
    waarin moet worden gezocht}
    MOVE 5 TO PRIJS
(1) FIND BESTELREGEL WITHIN ABR USING PRIJS
    ZOLANG ¬ EOS DOE
(2) | FIND OWNER WITHIN BBR
    | GET BESTELLING
    | PRINT(B#)
(3) | FIND DUPLICATE WITHIN ABR USING PRIJS
```

De setoccurrence van ABR waarin artikel A4 de owner is ziet er als volgt uit:



Hierin zijn de onvolledige setoccurrences van BBR met behulp van stippellijnen weergegeven.

Het verloop van de currency tabel is daarom als volgt:

	CRRU	CRRT BESTELREGEL	CRRT BESTELLING	CRST ABR	CRST BBR	
(1)	BR3	BR3	--	BR3	BR3	
(2)	B3	BR3	B3	BR3	B3	
(3)	BR4	BR4	B3	BR4	BR4	
(2)	B4	BR4	B4	BR4	B4	
(3)	B4	BR4	B4	BR4	B4	EOS

Afgedrukt wordt dus

B3

B4

10.8.6. Selectie van een al eerder geselecteerd record

Het formaat van deze opdracht is:

FIND data-base-key-identifier

Functie:
Het record dat geselecteerd wordt is het record waarvan de database key gelijk is aan de inhoud van de data-base-key-identifier.

Deze opdracht dient uitsluitend om de currency indicatoren te vullen met de inhoud van een hulpveld. In zo'n hulpveld is de inhoud van een currency indicator zoals die in een eerder stadium van het programma was bewaard.

De inhoud van een currency indicator kan worden bewaard in een zgn. data-base-key-identifier. Dit is niets anders dan een hulpveld in de UWA van het programma. Het veld kan worden gevuld met de opdracht

MOVE CURRENCY FOR { RUN-UNIT / recordnaam / setnaam } TO data-base-key-identifier

De inhoud van de gespecificeerde currency indicator wordt dan naar het aangegeven veld gebracht.

Deze opdrachten worden ondermeer gebruikt in situaties waarin een zoekproces moet worden voortgezet vanaf een eerder bereikt punt waar het werd afgebroken t.b.v. een ander zoekproces. Ook kunnen deze opdrachten worden gebruikt om een currency indicator die door een RETAINING werd bevroren te ontdooien.

Voorbeeld.

1. Maak een lijst van de artikelen waarvan bij fabriek F2 de voorraad onder de gewenste minimum voorraad is gedaald, met vermelding van de fabrieken waar deze artikelen ook worden geproduceerd.

```
MOVE F2 TO FAB#
FIND ANY FABRIEK USING FAB#
FIND FIRST VOORRAAD WITHIN FV
ZOLANG ¬ EOS DOE
┌─
│MOVE CURRENCY FOR FV TO HULP
│GET VOORRAAD
│IF VRD < MINVRD THEN
│ ┌─
│ │FIND OWNER WITHIN AV
│ │GET ARTIKEL
│ │PRINT(ART#)
│ │FIND FIRST VOORRAAD WITHIN AV
│ │ZOLANG ¬ EOS DOE
│ │ ┌─
│ │ │FIND OWNER WITHIN FV
│ │ │GET FABRIEK
│ │ │IF FAB# ≠ F2 THEN PRINT(FAB#)
│ │ │FIND NEXT VOORRAAD WITHIN AV
│ │ └─
│ └─
│FIND HULP
│FIND NEXT VOORRAAD WITHIN FV
└─
```

Afgedrukt wordt:

A3 F3

10.9. De STORE-opdracht

STORE recordnaam [RETAINING [{ RECORD / {setnaam}... }] CURRENCY]

Functie:

Het record van het opgegeven type wordt uit de UWA overgebracht naar de database. Is het recordtype member van een of meer settypen, dan wordt voor elk settype waarvoor INSERTION AUTOMATIC is gedeclareerd het nieuwe record ook als member in een setoccurrence opgenomen. Is INSERTION MANUAL gedeclareerd dan wordt het record niet in een setoccurrence opgenomen.

De setoccurrence waarin het record wordt opgenomen, wordt bepaald aan de hand van de voor de set gedeclareerde setselectie clausule. De grootheid die het systeem hiervoor nodig heeft moet dan de juiste waarde hebben. Afhankelijk van de specificatie in de setselectie clausule is deze grootheid:

by KEY: de sleutelwaarde van de owner van de gewenste setoccurrence;

by APPLICATION: de currency indicator van het betreffende settype.

Als gevolg van de STORE opdracht wordt de database key van het opgevoerde record in de currency indicator van de run-unit gezet. Verder worden bijgewerkt de currency indicator van het recordtype en de currency indicator van elk settype waarvan het record in een setoccurrence is opgenomen.

Voorbeelden.

1. Fabriek F4 gaat in het vervolg ook (het reeds voorkomende) artikel A3 produceren. De voorraad is 0 en de gewenste minium voorraad is 200. Neem deze gegevens op in de database.

```
MOVE F4 TO FAB#
MOVE A3 TO ART#
MOVE 0 TO VRD
MOVE 200 TO MINVRD
FIND ANY ARTIKEL USING ART#
STORE VOORRAAD
```

2. Indien een klant op een bepaalde datum een aantal artikelen bij een fabriek bestelt dan kunnen deze gegevens op de volgende manier in de database worden opgenomen.

   Voor de eenvoud veronderstellen we dat de gegevens van de bestellingen en de bestelde artikelen in een invoerbestand staan, dat een aanroep van de procedure LEES (BESTELLING) tot gevolg heeft dat de velden kb#, kklant, kdatum en kfab# worden gevuld met de gegevens van een bestelling en dat een aanroep van de procedure LEES (BESTELREGEL) tot gevolg heeft dat de velden kart, khoev en kprijs worden gevuld met de gegevens van het volgende bestelde artikel.

```
    LEES(BESTELLING)
    MOVE KB# TO B#
    MOVE KKLANT TO KLANT
    MOVE KDATUM TO DATUM
    MOVE KFAB# TO FAB#
(1) FIND ANY FABRIEK USING FAB#
(2) STORE BESTELLING
    LEES (BESTELREGEL)
    ZOLANG ¬ EOF DOE
    ┌
    │MOVE KART TO ART#
    │MOVE KHOEV TO HOEV
    │MOVE KPRIJS TO PRIJS
(3) │STORE BESTELREGEL
    │LEES (BESTELREGEL)
    └
```

Indien het invoerbestand de volgende gegevens bevat:

BESTELLING: B7, Jacobs, 22-10-1983, F1
BESTELREGEL1: A1, 150, 2.75
BESTELREGEL2: A5, 2000, 0.45

dan is het verloop van de currency indicatoren:

	CRRU	CRRT FABRIEK	CRRT BESTELLING	CRRT BESTELREGEL	CRST FB	CRST BBR	CRST ABR
(1)	F1	F1	null	null	F1	null	null
(2)	B7	F1	B7	null	B7	B7	null
(3)	BR1	F1	B7	BR1	B7	BR1	BR1
(3)	BR2	F1	B7	BR2	B7	BR2	BR2

10.10. De MODIFY-opdracht

Het formaat van deze opdracht is:
MODIFY recordnaam

Functie:
De inhoud van het record aangewezen door het current record van het record-type (CRRT recordnaam) wordt vervangen door de inhoud van het record in de UWA. Na afloop wordt het current record van het recordtype tevens current record van de run-unit (CRRU).

Voorbeelden.

1. Fabriek F1 is verouderd en wordt daarom afgebroken en opnieuw gebouwd in MOERDIJK

```
MOVE F1 TO FAB#
FIND ANY FABRIEK USING FAB#
GET FABRIEK
MOVE MOERDIJK TO PLAATS
MODIFY FABRIEK
```

2. De voorraad van artikel A3 in fabriek F2 wordt met 40 eenheden verhoogd.

```
MOVE F2 TO FAB#
FIND ANY FABRIEK USING FAB#
FIND FIRST VOORRAAD WITHIN FV
GET VOORRAAD
FIND OWNER WITHIN AV
GET ARTIKEL
ZOLANG ART# ≠ A3 DOE
┌
│FIND NEXT VOORRAAD WITHIN FV
│GET VOORRAAD
│FIND OWNER WITHIN AV
│GET ARTIKEL
└
ADD 40 TO VRD
MODIFY VOORRAAD
```

10.11. De ERASE-opdracht

Het formaat van deze opdracht is:

ERASE [ALL] recordnaam

Functie:

Het objectrecord is het current record van het opgegeven recordtype (CRRT recordnaam).

Dit record wordt uit de database verwijderd indien het geen owner is van een set of indien de setoccurrence leeg is. In het andere geval hangt de werking van de opdracht af van het al of niet opgeven van ALL.

Is ALL niet gespecificeerd, dan kan een setoccurrence uitsluitend worden verwijderd indien het lidmaatschap van het membertype fixed of optional is. Fixed members worden verwijderd, terwijl optional members worden losgekoppeld uit de setoccurrence.

Is ALL wel gespecificeerd dan wordt de betreffende setoccurrence in zijn geheel verwijderd.

In alle gevallen geldt dat elk verwijderd record op zijn beurt weer wordt beschouwd als objectrecord. Op deze wijze kan met één ERASE-opdracht een "boom" van records worden verwijderd.

Voorbeelden.

1. Fabriek F5 wordt uit de database verwijderd.

```
MOVE F5 TO FAB#
FIND ANY FABRIEK USING FAB#
ERASE FABRIEK
```

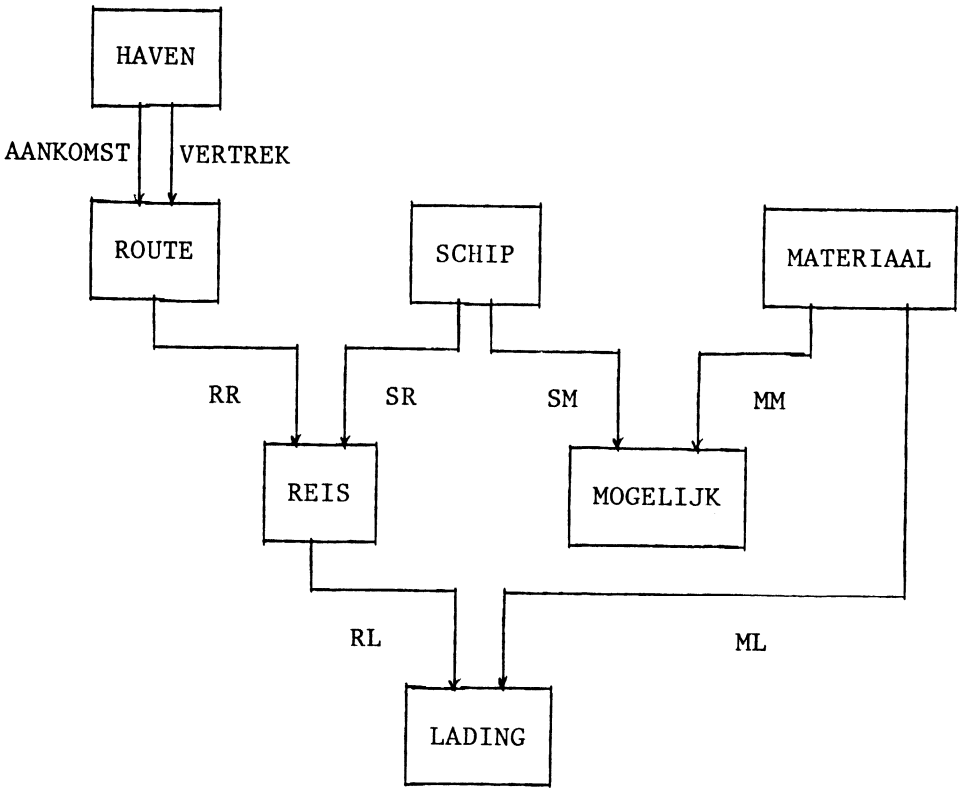
2. Bestelling B1 is volledig uitgevoerd en kan derhalve uit de database worden verwijderd.

```
MOVE B1 TO B#
FIND ANY BESTELLING USING B#
ERASE ALL BESTELLING
```

10.12. Vraagstukken

1. Rederij

Een rederij zet een database op voor de planning van de door haar schepen te varen routes en de te vervoeren ladingen.
Een route is een rechtstreekse verbinding tussen een vertrek- en aankomsthaven. Van een haven wordt het liggeld per dag opgenomen. Ieder schip vaart een route in een aantal dagen dat ligt tussen een zeker minimum en maximum aantal dagen.
De rederij beperkt zich tot het vervoer van een bepaald pakket materialen waarover gegevens zijn vastgelegd. Van de schepen wordt vastgelegd de naam, het tonnage en de materialen die het in principe kan vervoeren.
De schepen vervoeren op een reis ladingen. Van een reis worden de geplande vertrek- en aankomstdatum vastgelegd en de hoeveelheden te vervoeren materialen. Aangenomen wordt dat op een reis eventueel andere materialen worden vervoerd dan die waarvoor het schip bestemd is. Het netwerkmodel voor deze database is:



Het schema van deze database luidt:

```
SCHEMA REDERIJ


RECORD HAVEN
    KEY HN IS HNAAM DUPLICATES NOT ALLOWED
    HNAAM
    LIGGELD

RECORD ROUTE
    MIN-VAARTIJD
    MAX-VAARTIJD

RECORD SCHIP
    KEY SN IS SNAAM DUPLICATES NOT ALLOWED
    SNAAM
    TONNAGE

RECORD REIS
    VDATUM
    ADATUM

RECORD MATERIAAL
    KEY MNR IS M# DUPLICATES NOT ALLOWED
    M#
    MNAAM
```

```
RECORD MOGELIJK

RECORD LADING
    HOEV

SET AANKOMST
    OWNER HAVEN
    ORDER FIRST
    MEMBER ROUTE INSERTION AUTOMATIC RETENTION FIXED
    SET SELECTION BIJ KEY HN

SET VERTREK
    OWNER HAVEN
    ORDER FIRST
    MEMBER ROUTE INSERTION AUTOMATIC RETENTION FIXED
    SET SELECTION BY APPLICATION

SET RR
    OWNER ROUTE
    ORDER SORTED
    MEMBER REIS INSERTION AUTOMATIC RETENTION FIXED
    KEY IS ASCENDING VDATUM
    SET SELECTION BY APPLICATION

SET SR
    OWNER SCHIP
    ORDER SORTED
    MEMBER REIS INSERTION AUTOMATIC RETENTION FIXED
    KEY IS ASCENDING VDATUM DUPLICATES NOT ALLOWED
    SET SELECTION BY KEY SN

SET SM
    OWNER SCHIP
    ORDER LAST
    MEMBER MOGELIJK INSERTION AUTOMATIC RETENTION FIXED
    SET SELECTION BY APPLICATION

SET MM
    OWNER MATERIAAL
    ORDER LAST
    MEMBER MOGELIJK INSERTION AUTOMATIC RETENTION FIXED
    SET SELECTION BY APPLICATION

SET RL
    OWNER REIS
    ORDER FIRST
    MEMBER LADING INSERTION AUTOMATIC RETENTION FIXED
    SET SELECTION BY APPLICATION

SET ML
    OWNER MATERIAAL
    ORDER FIRST
    MEMBER LADING INSERTION AUTOMATIC RETENTION FIXED
    SET SELECTION BY KEY MNR
```

Schrijf DML-progrmama's waarmee de volgende gegevens worden opgevoerd c.q. verkregen. Er kan van worden uitgegaan dat de in de vragen genoemde record-occurrences in de database aanwezig zijn. Aan de opmaak van de overzichten worden geen eisen gesteld. Zo mogen opeenvolgende regels gedeeltelijk dezelfde gegevens bevatten.

N.B. Bij de beantwoording van de vragen mag geen gebruik worden gemaakt van array's.

1. Maak een lijst van de materialen die de Neeltje Jacoba kan vervoeren.

2. Maak een lijst van de schepen die op 31-03-1984 vertrekken.

3. Maak een overzicht van de geplande reizen van de Rijndam. Vermeld hierin vertrekhaven, aankomsthaven, vertrekdatum en aankomstdatum.

4. Maak een lijst van de schepen die op 1-4-1984 van Rotterdam vertrekken. Vermeld hierin de aankomstdatum en de haven van bestemming.

5. Maak een lijst van de schepen die op 1-5-1984 in New York aankomen. Vermeld hierin de vertrekhaven en de vertrekdatum.

6. Maak een lijst van de lading (materiaal en hoeveelheid) die de Neptunus zal vervoeren op de reis van Rotterdam naar New York met vertrekdatum 8-7-1984.

7. Maak een lijst van de schepen waarvoor een reis van Rotterdam naar New York is gepland. De vertrek- en aankomstdatum moeten worden afgedrukt.

8. Maak een overzicht van de materialen die de Rijndam op de geplande reis met vertrekdatum 3-4-1984 zal gaan vervoeren. Vermeld per materiaal de schepen die dit materiaal in principe ook zouden kunnen vervoeren.

9. De Rijndam heeft zodanige schade opgelopen dat alle voor haar geplande reizen moeten vervallen. Om na te gaan welke schepen het vervoer kunnen overnemen, moet een overzicht worden gemaakt van de schepen waarvoor reizen zijn gepland op routes waarop de Rijndam is ingezet. Een voor-

beeld van zo'n overzicht:

VAN	NAAR	VDAT1	SCHIP	VDAT2
R'DAM	NEW YORK	080484	AEGIR	120484
R'DAM	NEW YORK	080484	NEPTUNUS	130484
NEW YORK	DAKAR	200484	ZEELEEUW	180484
NEW YORK	DAKAR	200484	VOLHARDING	260484
NEW YORK	DAKAR	200484	AEGIR	010584
....	....	....	....	....

10. De rederij neemt een nieuw schip in de vaart. De gegevens van het nieuwe schip en de nummers van de materialen die dit schip kan vervoeren staan in een invoerbestand. De gegevens van het nieuwe schip worden ingelezen in KSNAAM en KTONNAGE met de opdracht LEES(SCHIP). Een materiaalnummer wordt ingelezen in KM met de opdracht LEES(MATERIAAL). Aangenomen mag worden dat de gegevens over de materialen al in de database aanwezig zijn.
    Gevraagd wordt een programma voor het opslaan van de nieuwe gegevens.

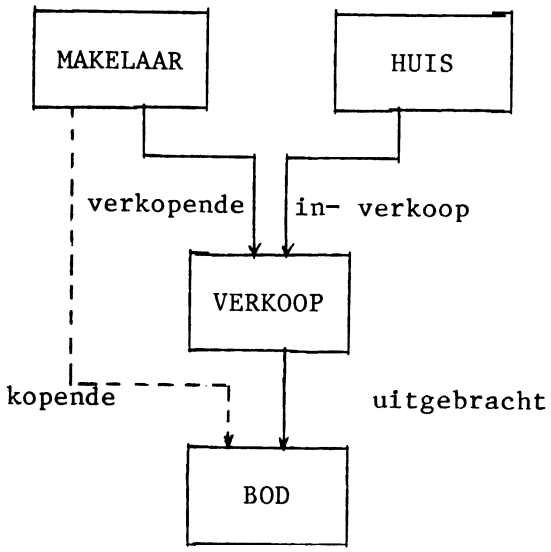
11. De rederij gaat op een nieuwe route van Amsterdam naar Bangkok varen. De minimum vaartijd bedraagt 6 weken en de maximum vaartijd 8 weken. Neem deze gegevens op in de database.

2. **Makelaars**

De bond van makelaars besluit ten behoeve van de koop en verkoop van huizen een database te gaan gebruiken. Van makelaars wordt de kantoornaam (uniek), het kantooradres, de naam van de makelaar en het telefoonnummer opgenomen. Makelaars ontvangen van eigenaars van huizen gedateerde opdrachten tot verkoop. Een eigenaar kan één of meer andere makelaars opdracht tot verkoop geven indien de verkoop van het huis te lang gaat duren. Ook kan dan de vraagprijs worden verlaagd.
Kopers kunnen een bod uitbrengen op een huis bij één van de verkopende make-

laars. Zij kunnen dit bod zelf uitbrengen of dat laten doen via de door hen ingeschakelde makelaar.
De verdeling van de courtage gebeurt aan de hand van de kosten die de makelaars hebben gemaakt.

Het netwerkmodel voor deze database is:



Het schema van deze database is:

```
SCHEMA MAKELAARS


RECORD MAKELAAR
   KEY KNAAM IS KANTOORNAAM DUPLICATES NOT ALLOWED
   KANTOORNAAM
   KANTOORADRES
   NAAM
   TELEFOON

RECORD HUIS
   KEY HUIS-IDENT IS PLAATS,STRAAT,HUISNR DUPLICATES NOT ALLOWED
   PLAATS
   STRAAT
   HUISNR
   EIGENAAR
   VRAAGPRIJS

RECORD VERKOOP
   VDATUM
   KOSTEN

RECORD BOD
   BDATUM
   BEDRAG
   KOPER

SET VERKOPENDE
   OWNER MAKELAAR
   ORDER FIRST
   MEMBER VERKOOP INSERTION AUTOMATIC RETENTION FIXED
   SET SELECTION BY KEY KNAAM

SET IN-VERKOOP
   OWNER HUIS
   ORDER FIRST
   MEMBER VERKOOP INSERTION AUTOMATIC RETENTION FIXED
   SET SELECTION BY APPLICATION

SET UITGEBRACHT
   OWNER VERKOOP
   ORDER SORTED
   MEMBER BOD INSERTION AUTOMATIC RETENTION FIXED
   KEY IS ASCENDING BDATUM DUPLICATES ARE ALLOWED
   SET SELECTION BY APPLICATION

SET KOPENDE
   OWNER MAKELAAR
   ORDER FIRST
   MEMBER BOD INSERTION MANUAL RETENTION FIXED
   SET SELECTION BY KEY KNAAM
```

Schrijf DML programma's waarmee de volgende gegevens worden opgevoerd c.q. verkregen. Er kan van worden uitgegaan dat de in de vragen genoemde record occurrences in de database aanwezig zijn. Aan de opmaak van de overzichten worden geen eisen gesteld. Zo mogen opeenvolgende regels gedeeltelijk dezelfde gegevens bevatten.

1. De eigenaar van het huis Kerklaan 25 te Toren geeft omdat de verkoop te lang gaat duren, op 21 juni 1983 makelaar Vastgoed ook een opdracht tot verkoop van het huis. Dezelfde dag nog brengt koper Jansen bij deze makelaar een bod uit op het huis van fl. 100.000,--.

2. Makelaar Losgoed wil een overzicht van de huizen waarop hij namens een koper een bod heeft uitgebracht. In dit overzicht moeten per bod de koper, het bedrag, het huis en de verkopende makelaar worden afgedrukt.

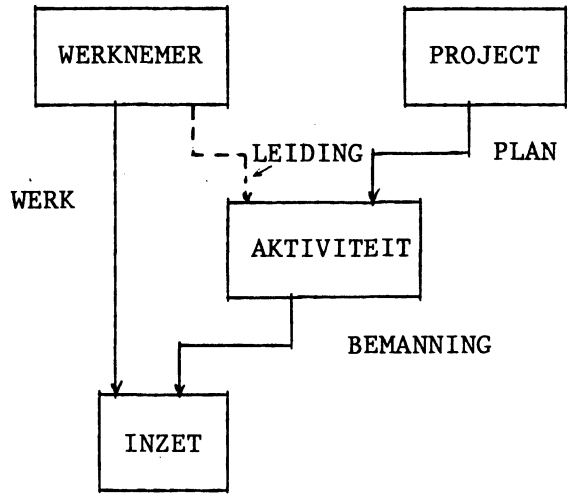
3. Een overzicht van de huizen die bij makelaar Vastgoed in de verkoop staan, en bij elk huis alle andere makelaars waarbij het in de verkoop staat.

4. Het bedrag en de datum van het laatste bod dat koper Koopgraag op het huis Kerklaan 25 te Toren bij makelaar Losgoed heeft uitgebracht. Indien hij dit bod heeft laten uitbrengen dan ook de naam van de betreffende makelaar.

3. Projectuitvoering

Een bedrijf voert projecten uit. Ieder project omvat een aantal aktiviteiten zoals analyse, ontwerp, bouw, enz. De toestand waarin een aktiviteit verkeert wordt aangegeven met 'P' (gepland), 'S' (gestart) of 'E' (beëindigd). Werknemers kunnen worden ingezet voor één of meer aktiviteiten. Van een inzetbare werknemer wordt het aantal uren dat hij kan worden ingezet voor een aktiviteit in de database opgenomen (dus uren-inzetbaar > 0). Voordat een aktiviteit wordt gestart, wordt een leider aangewezen.

Het netwerkmodel van deze database luidt:



Het bijbehorende schema luidt:

```
SCHEMA PROJECT-ADMINISTRATIE


RECORD WERKNEMER
   KEY WKN-KEY IS PERS-NUMMER DUPLICATES NOT ALLOWED
   PERS-NUMMER
   PERS-NAAM
   FUNKTIE

RECORD INZET
   UREN-INZETBAAR
   UREN-BESTEED

RECORD PROJECT
   KEY PRJ-KEY IS PROJ-NUMMER DUPLICATES NOT ALLOWED
   PROJ-NUMMER
   PROJ-NAAM
   PROJ-OMS

RECORD AKTIVITEIT
   AKT-NUMMER
   BEGINDATUM
   TOESTAND

SET WERK
   OWNER WERKNEMER
   ORDER SORTED
   MEMBER INZET INSERTION AUTOMATIC RETENTION MANDATORY
   KEY IS ASCENDING UREN-INZETBAAR DUPLICATES ALLOWED
   SET SELECTION BY APPLICATION
```

```
SET LEIDING
   OWNER WERKNEMER
   ORDER LAST
   MEMBER AKTIVITEIT INSERTION MANUAL RETENTION OPTIONAL
   SET SELECTION BY APPLICATION

SET PLAN
   OWNER PROJECT
   ORDER SORTED
   MEMBER AKTIVITEIT INSERTION AUTOMATIC RETENTION MANDATORY
   KEY IS ASCENDING AKT-NUMMER DUPLICATES NOT ALLOWED
   SET SELECTION BY KEY PRJ-KEY

SET BEMANNING
   OWNER AKTIVITEIT
   ORDER FIRST
   MEMBER INZET INSERTION AUTOMATIC RETENTION MANDATORY
   SET SELECTION BY APPLICATION
```

Schrijf DML programma's waarmee de volgende gegevens worden opgevoerd c.q. verkregen. Er kan van worden uitgegaan dat de in de vragen genoemde record occurrences in de database aanwezig zijn. Aan de opmaak van de overzichten worden geen eisen gesteld. Zo mogen opeenvolgende regels gedeeltelijk dezelfde gegevens bevatten.

1. Een overzicht van de inzet van werknemer 1837. Per aktiviteit moet het aantal inzetbare uren, het aantal bestede uren, de aktiviteitleider en de projectnaam worden afgedrukt.

2. Daar de voortgang van aktiviteit 103 van project 300100 in gevaar komt wordt besloten dat de werknemers 1012 en 873 ook worden ingezet voor resp. 40 en 60 uren.

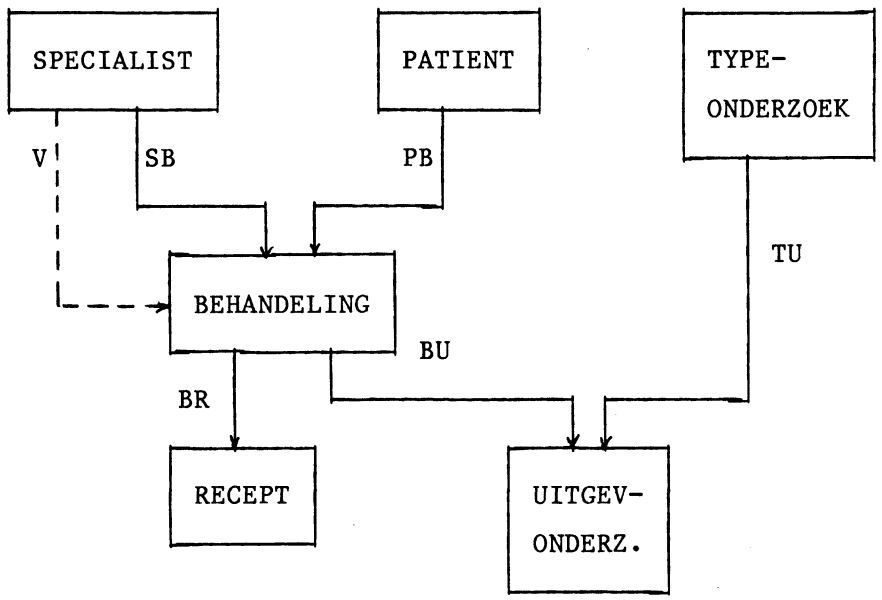
3. Een overzicht van de gestarte aktiviteiten waarvan werknemer 1937 de leiding heeft. Per aktiviteit moet worden afgedrukt het project en de namen van de werknemers die voor deze aktiviteit worden ingezet.

4. Een overzicht van de beëindigde aktiviteiten van het project 310200. Per aktiviteit moet worden afgedrukt het aktiviteitnummer, de naam en het aantal bestede uren van de aktiviteitleider.

4. Ziekenhuis

Een ziekenhuis gebruikt voor de administratie van poliklinische behandelingen van patienten een database. Het ziekenhuis registreert van patienten NAW-gegevens, geboortedatum en geslacht. Een patient komt onder behandeling van één of meer specialisten. Iedere specialist heeft zijn eigen specialisme. Gedurende de periode van behandeling kan de specialist verschillende onderzoeken voorschrijven. De verschillende mogelijke onderzoeken (b.v. ECG, darmfoto, bloedonderzoek, etc.) zijn gecodeerd en voorzien van een naam. Een patient krijgt voordat een onderzoek wordt uitgevoerd een beschrijving ervan.
Tijdens een behandeling kunnen medicijnen worden voorgeschreven. Voor ieder medicijn afzonderlijk wordt een recept verstrekt waarop de dosis is vermeld. De data van de tijdens de behandeling verrichte onderzoeken en verstrekte recepten moeten kunnen worden opgevraagd.
Het is mogelijk dat een specialist voor een behandeling één collega inschakelt als vervanger.

Het netwerkmodel van de ziekenhuis database luidt:



Het bijbehorende schema luidt:

```
SCHEMA ZIEKENHUIS


RECORD SPECIALIST
   KEY SPECNR IS S# DUPLICATES NOT ALLOWED
   S#
   SNAAM
   SPECIALISME

RECORD PATIENT
   KEY PATNR IS P# DUPLICATES NOT ALLOWED
   P#
   PNAAM
   GEBDAT
   ADRES
   WOONPLAATS

RECORD BEHANDELING
   BEGINDATUM

RECORD RECEPT
   DATUM
   NAAM MEDICIJN
   DOSIS

RECORD TYPE-ONDERZOEK
   KEY ONDNR IS O# DUPLICATES NOT ALLOWED
   O#
   ONAAM
   BESCHRIJVING

RECORD UITGEV-ONDERZOEK
   DATUM

SET SB
   OWNER SPECIALIST
   ORDER LAST
   MEMBER BEHANDELING INSERTION AUTOMATIC RETENTION MANDATORY
   SET SELECTION BY KEY SPECNR

SET PB
   OWNER PATIENT
   ORDER LAST
   MEMBER BEHANDELING INSERTION AUTOMATIC RETENTION MANDATORY
   SET SELECTION BY APPLICATION

SET V
   OWNER SPECIALIST
   ORDER LAST
   MEMBER BEHANDELING INSERTION MANUAL RETENTION OPTIONAL
   SET SELECTION BY APPLICATION

SET BR
   OWNER BEHANDELING
   ORDER LAST
   MEMBER RECEPT INSERTION AUTOMATIC RETENTION MANDATORY
   SET SELECTION BY APPLICATION
```

```
SET BU
   OWNER BEHANDELING
   ORDER LAST
   MEMBER UITGEV-ONDERZOEK INSERTION AUTOMATIC RETENTION MANDATORY
   SET SELECTION BY APPLICATION

SET TU
   OWNER TYPE-ONDERZOEK
   ORDER SORTED
   MEMBER UITGEV-ONDERZOEK INSERTION AUTOMATIC RETENTION MANDATORY
   KEY IS ASCENDING DATUM DUPLICATES ALLOWED
   SET SELECTION BY KEY ONDNR
```

Schrijf DML programma's waarmee de volgende gegevens worden opgevoerd c.q. verkregen. Er kan van worden uitgegaan dat de in de vragen genoemde record-occurrences in de database aanwezig zijn. Aan de opmaak van de overzichten worden geen eisen gesteld. Zo mogen opeenvolgende regels gedeeltelijk dezelfde gegevens bevatten.

1. Patient Bos (P# = 37) komt vanwege nieuwe klachten onder behandeling bij specialist Jong (S# = 14).
   Tijdens het eerste consult op 16-02-84 schrijft deze specialist een recept uit voor het medicijn xy in dosis 20.

2. Maak een overzicht van de patienten waarvan op 1-8-1983 een maagfoto (O# = 21) is gemaakt.
   Per patient moeten eerst de naam en de geb.datum worden afgedrukt en daarna alle onderzoeken (datum en naam) die na 1-8-83 zijn verricht.

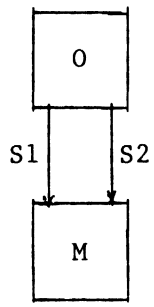
N.B. Aangezien het aantal gemaakte maagfoto's zeer groot is, moet een DML programma worden gemaakt waarin niet alle memberoccurrences van de set TU worden bekeken.

3. Maak een overzicht van de patienten die door specialist Jacobse (S# = 33) worden behandeld. Vermeld per patient naam, adres en evt. de naam van de specialist die als vervanger is opgetreden.

4. Maak een overzicht van de recepten die patient Bos (P# = 37) heeft gekregen voor de behandeling die gestart is op 27-5-1983.

5. Currency en occurrence

Gegeven is het volgende netwerkmodel met bijbehorend schema.



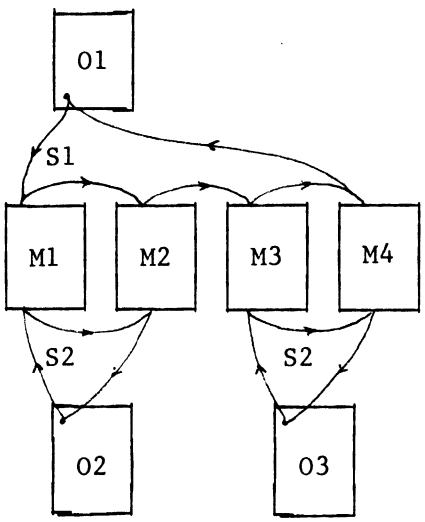

```
RECORD O
 KEY ONR IS O# DUPLICATES NOT ALLOWED
 O#

RECORD M
 M#


SET S1
 OWNER O
 ORDER SORTED
 MEMBER M INSERTION AUTOMATIC RETENTION FIXED
 KEY IS ASCENDING M# DUPLICATES ALLOWED
 SET SELECTION BY KEY ONR

SET S2
 OWNER O
 ORDER LAST
 MEMBER M INSERTION AUTOMATIC RETENTION FIXED
 SET SELECTION BY APPLICATION
```

De database bevat op zeker ogenblik de volgende record- en setoccurrences:



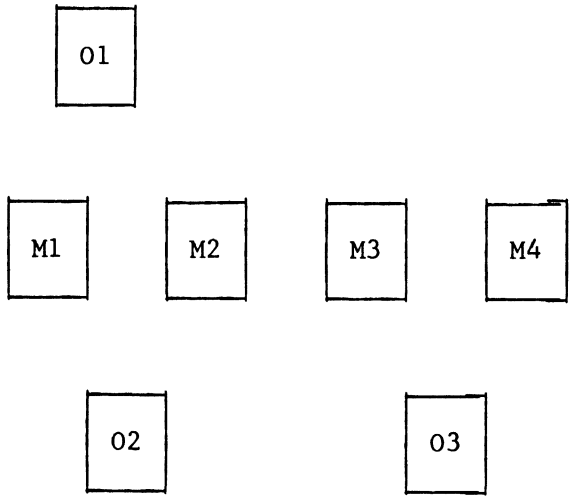
a. Op deze database worden de volgende DML-opdrachten achtereenvolgens uitgevoerd.

```
    O#:=O1
    M#:=M2
1   FIND M WITHIN S1 USING M# RETAINING M
    M#:=M3
2   FIND M WITHIN S2 USING M#
    M#:=M4
3   FIND ANY M USING M#
    O#:=O4
4   STORE O RETAINING S2
    M#:=M5
5   STORE M RETAINING S1
6   FIND OWNER WITHIN S1
```

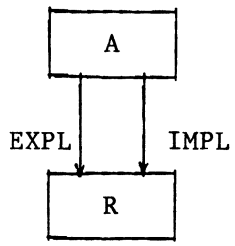
Geef de inhoud van de currency registers na afloop van elke DML-opdracht. Gebruik hiervoor de volgende currency tabel waarin de beginstand van de currency registers is vermeld.

	CRRU	CRRT O	CRRT M	CRST S1	CRST S2
beginstand	O2	O2	M1	O2	M4
1					
2					
3					
4					
5					
6					

b. Teken de record- en setoccurrences die de database bevat na afloop van de laatste DML-opdracht. Geef de namen van de setoccurrences aan. Geef door pijlen de volgorde van de memberoccurrences aan.



2. Gegeven is het volgende netwerkmodel met bijbehorend schema.



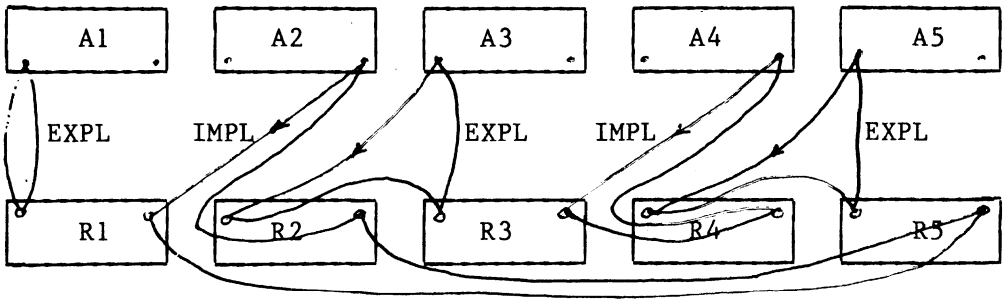

```
RECORD A
   KEY ANR IS A# DUPLICATES NOT ALLOWED
   A#

RECORD R
   R#

SET EXPL
   OWNER A
   MEMBER R
   ...

SET IMPL
   OWNER A
   MEMBER R
   ...
```

De database bevat op zeker ogenblik de volgende record- en setoccurrences:



Op deze database worden de onderstaande DML-opdrachten achtereenvolgens uitgevoerd.

```
A# := A5
1  FIND ANY A USING A#
2  FIND NEXT RECORD WITHIN EXPL
3  FIND OWNER WITHIN IMPL RETAINING EXPL
4  FIND NEXT RECORD WITHIN EXPL
5  FIND OWNER WITHIN IMPL
A# := A3
6  FIND ANY A USING A#
7  FIND NEXT RECORD WITHIN EXPL
```

Geef de inhoud van de currency registers aan na afloop van elke DML-opdracht. Gebruik daartoe de onderstaande tabel.

	CRRU	CRRT A	CRST EXPL	CRST IMPL
1				
2				
3				
4				
5				
6				
7				