KU Leuven | Campus De Nayer
Industrieel ingenieur
Opleiding chemie
3e academisch bachelorjaar

Databanken

Academiejaar 2019-20

J. Vennekens
H. Crauwels

Inhoudsopgave

1 Database management systemen. 1
- 1.1 Objectieven . . . 1
- 1.2 Schema's . . . 2
- 1.3 Data modellen . . . 3
- 1.4 Data definitie taal en data manipulatie taal . . . 7
- 1.5 Database administrator . . . 8
- 1.6 De architectuur van een DBMS . . . 8
- 1.7 Client-server architectuur . . . 10

2 Analyse van gegevens: entity-relationship model 14
- 2.1 Entiteiten en entiteit-verzamelingen . . . 14
- 2.2 Attributen . . . 14
- 2.3 Relaties en relatie-verzamelingen . . . 15
- 2.4 Integriteitsbeperkingen . . . 15
- 2.5 Primaire sleutels . . . 16
- 2.6 ER-diagram . . . 17
- 2.7 Herleiden van ER-diagrammen naar tabellen . . . 19
- 2.8 Generalisatie en specialisatie . . . 20
- 2.9 Aggregatie . . . 22
- 2.10 Voorbeelden . . . 23

3 SQL: Data Definition/Manipulation Language 26
- 3.1 Voorbeeld van een eenvoudige databank . . . 26
- 3.2 Maken van nieuwe tabellen . . . 26
- 3.3 Verwijderen van tabellen . . . 27
- 3.4 Beperkingen . . . 27
- 3.5 Invoeren, wijzigen en verwijderen van gegevens. . . . 29
- 3.6 Inhoud van de tabellen . . . 30

4 SQL: het vraagtaal gedeelte 32
- 4.1 Componenten van de SELECT-instructie . . . 32
- 4.2 De FROM-component . . . 33
- 4.3 De WHERE-component . . . 33
- 4.4 Gegevens uit meerdere tabellen . . . 36
- 4.5 GROUP BY en HAVING . . . 37
- 4.6 De subquery . . . 40
- 4.7 Combineren van select-instructies . . . 44
- 4.8 Select-instructie : de join . . . 45

5 Online analytical processing 48
- 5.1 Definities . . . 48
- 5.2 Een multi-dimensioneel model . . . 49
- 5.3 Aggregatie . . . 51
- 5.4 Hiërarchiën . . . 53
- 5.5 Implementatie issues . . . 55
- 5.6 Laden van een data warehouse . . . 55

6 Information retrieval 57
- 6.1 Situering . . . 57
- 6.2 Indexing . . . 58
- 6.3 Gelijkaardigheid en relevantie . . . 60
- 6.4 Oracle Text . . . 63

A Oracle SQL*Plus **68**

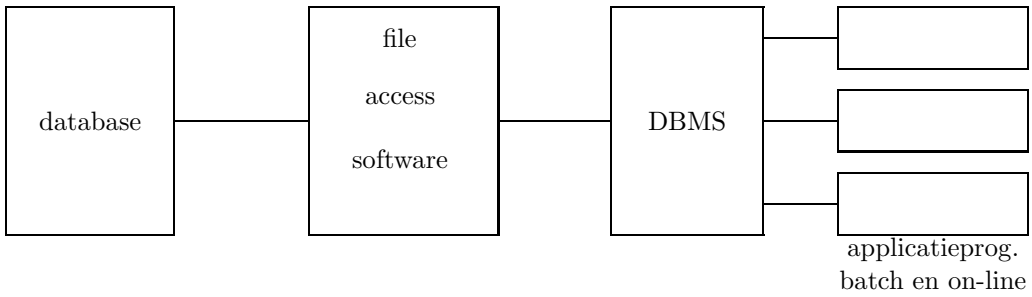
B Een databasetoepassing **69**
B.1 Omschrijving van de gegevens 69
B.2 Opgaven voor de practicumzittingen 69

1 Database management systemen.

1.1 Objectieven

Een *database management systeem* (DBMS) bestaat uit een verzameling inter-gerelateerde data en een verzameling programma's om toegang te hebben tot deze data. De verzameling gegevens wordt meestal *database* genoemd. De belangrijkste doelstelling van een DBMS is een omgeving te realiseren waarin men informatie kan opvragen en aanpassen in de database op een zo *gemakkelijk* en *efficiënt* mogelijke manier.

Meestal bestaan er verschillende relaties tussen de verschillende verzamelingen gegevens in een organisatie. Bijvoorbeeld bij de verzameling leveranciers en de verzameling klanten kan het zijn dat een bepaalde klant ook leverancier is. Wanneer zijn adres op de twee plaatsen gestockeerd wordt, is dit redundantie en kan dit leiden tot tegenstrijdigheden. Wanneer deze gegevens centraal op één plaats gestockeerd worden, spreekt men van een *database* of gegevensbank.
De verschillende applicaties refereren allen naar één en dezelfde database. Dit gebeurt via een gemeenschappelijk software blok voor de file toegang, waarin de sequentiële of directe bestandsorganisatie geimplementeerd wordt.
Voorbeeld: een applicatie verwerkt een aantal met elkaar verbonden dataelementen uit records die zich in verschillende files bevinden. Hiervoor moeten de indextabellen geraadpleegd worden om de fysische lokaties van de verschillende records in de verschillende files te vinden. Dan kunnen deze records gelezen worden en kunnen de dataelementen uit deze records gehaald worden. Dus in de programmatuur van zo'n applicatie wordt gebruik gemaakt van fysische lokaties en van bestands- en recordstructuren. Wanneer een nieuw dataelement wordt toegevoegd aan een record, wat dus een nieuwe record structuur geeft, moeten alle programma's die dit record gebruiken aangepast worden; zelfs deze waarvoor het nieuwe dataelement irrelevant is.
Ideaal gezien zouden applicatieprogramma's moeten in staat zijn toegang te krijgen tot gegevens in een database onafhankelijk van de fysische structuur van de gegevensbank. Dus een programma zou moeten in staat zijn enkel die dataelementen te vragen en te krijgen van de databank, zonder daarvoor de gehele verzameling van records waarin deze dataelementen zitten te moeten verwerken.
Om dit te realiseren worden sinds de beginjaren '70 *database management systemen* (DBMS) ontwikkeld. Een DBMS is een bijkomende software laag tussen de file toegang software en de applicatieprogramma's, zoals voorgesteld in figuur 1.



Figuur 1: De verschillende toegangen tot de databank

Definitie van een databank volgens C. Date:

> A database is a computerized system whose overall purpose is to *maintain* information and to make that information *available* on demand.

De rol van het DBMS:

1. Het organiseren van de gegevens volgens de globale logische structuur (het schema) en de fysische opslag.

2. Toegang verschaffen aan de verschillende applicatieprogramma's zodat voldaan wordt aan de verschillende vereisten van de gebruikers (de subschema's = logische structuur van de database zoals de gebruiker die nodig heeft).

Een belangrijke bijkomende faciliteit van een DBMS pakket is een vraagtaal (query language). Dit is een voorbeeld van een vierde generatietaal: een verzameling gemakkelijk te gebruiken computerinstructies. Deze laten een computerleek toe om een specifiek dataelement uit de database op te vragen, toe te voegen, te wijzigen of te verwijderen.
Voordelen van een DBMS:

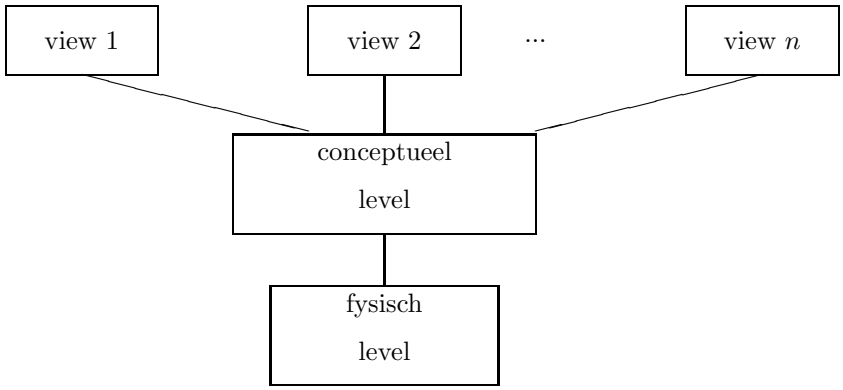
Gegevensintegratie: men vermijdt duplicatie en dus ook tegenstrijdigheden in de gegevens. Door de gegevens slechts eenmaal te stockeren wordt een minimum aan geheugencapaciteit gebruikt en wordt ook de onderhoudskost gereduceerd.

Gegevensonafhankelijkheid: d.m.v. de scheiding tussen de fysisch gestockeerde gegevensbestanden en de programma's die gebruik maken van deze gegevensbestanden. Dit laat aan de verschillende toepassingen toe om een andere visie te hebben op dezelfde gegevens. En bij wijzigingen in de database ondervinden de applicatieprogramma's daar niet de minste hinder van.

Gegevensintegriteit: omwille van de gecentraliseerde controle over deze gegevens (de functie van de database administratie): garanderen van bepaalde beperkingen of condities en de beveiliging (wie welke gegevens kan raadplegen, toevoegen, wijzigen, verwijderen).

1.2 Schema's

Databases veranderen gedurig omdat informatie toegevoegd en verwijderd wordt. De verzameling informatie die op een bepaald ogenblik in de database aanwezig is, wordt een *instance* van de database genoemd. Het globale ontwerp van de database wordt *schema* genoemd. Schema's veranderen zeldzaam, soms zelfs niet. De concepten schema en instance kunnen vergeleken worden met type definitie en variabele deklaratie in een programmeertaal.
Er bestaan verschillende schema's in een database: op het laagste niveau van abstractie vinden we het *fysisch* schema terug; op het intermediaire niveau hebben we het *conceptuele* schema; en op het hoogste niveau zijn verschillende *subschema's* gedefiniëerd.



Figuur 2: De drie niveau's van data abstractie

Data abstractie. De database kan benaderd worden vanuit verschillende levels van abstractie (figuur 2):

- fysisch level: laagste niveau van abstractie: hier wordt aangegeven *hoe* de data in wezen gestockeerd is door middel van complexe, gedetailleerde data structuren;

- conceptueel level: beschrijving van *welke* data in de database aanwezig is en welke relaties tussen deze informatie bestaat door middel van een klein aantal relatief eenvoudige structuren; dit is het level waarop de database administrator werkt, de persoon die beslist welke informatie in de database opgenomen wordt;
- view level: beschrijving van slechts een deel van de database; omwille van de omvang van de gehele database kan het conceptuele level nog vrij complex zijn; veel gebruikers zijn echter niet geinteresseerd in alle informatie die in de database gestockeerd is, maar slechts in een gedeelte; om de interactie van deze gebruikers met het systeem te vergemakkelijken wordt per specifieke gebruiker een view level gedefinieëerd.

Data onafhankelijkheid. Zoals reeds aangehaald zijn er drie niveau's van abstractie. De mogelijkheid om een schema definitie in één niveau aan te passen zonder daarbij effect te hebben op de schema definitie van een niveau hoger, wordt *data onafhankelijkheid* genoemd.

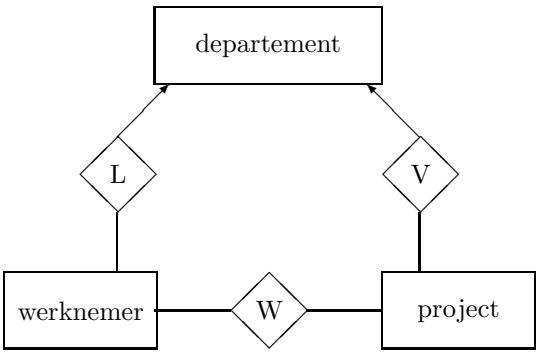
- Fysische data onafhankelijkheid: de mogelijkheid om het fysisch schema aan te passen zonder dat daarbij de applicatie programma's moeten herschreven worden. Deze aanpassingen zijn soms nodig om de performantie te verbeteren.
- Logische data onafhankelijkheid: de mogelijkheid om het logische schema aan te passen zonder dat daarbij de applicatie programma's moeten herschreven worden. Deze aanpassingen zijn nodig telkens de logische structuur van de database verandert bijvoorbeeld door toevoegingen van nieuwe informatie elementen.

Logische data onafhankelijkheid is moeilijker te verwezenlijken dan fysische data onafhankelijkheid: applicatie programma's zijn meestal sterk afhankelijk van de logische structuur van de gegevens.

1.3 Data modellen

Om de structuur van een database te beschrijven, gebruikt men het concept van het *data model*. Een data model is een verzameling conceptuele hulpmiddelen voor het beschrijven van de gegevens, de relaties tussen deze gegevens, de semantiek van de gegevens en de beperkingen op deze gegevens. Er zijn drie invalshoeken: het object gebaseerde logische model, het record gebaseerd logisch model en het fysisch gegevens model. De twee eerste modellen worden gebruikt om de gegevens te beschrijven op het conceptuele en view level, terwijl het laatste gebruikt wordt voor de beschrijving van de gegevens op het laagste abstractie niveau.

1.3.1 Object gebaseerde modellen



In naaststaand ER-diagram wordt weergegeven dat elke werknemer lid is (L) van één departement en aan verschillende projecten kan meewerken (W).
Elk project wordt binnen één departement uitgevoerd (V).

Figuur 3: Entity-Relationship diagram

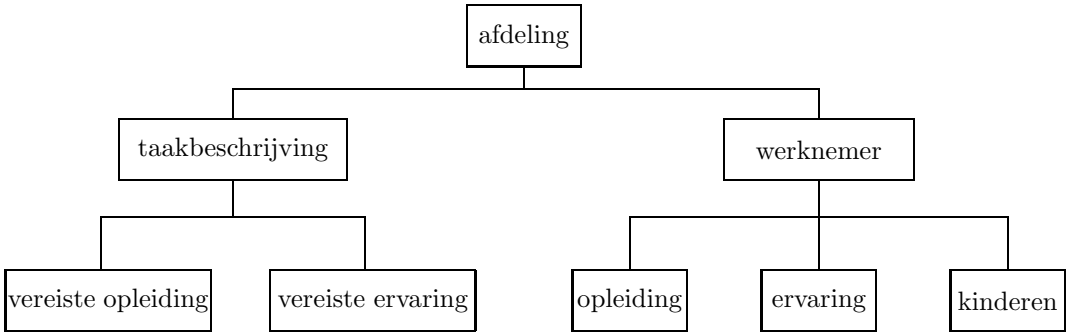
Deze modellen beschikken over vrij flexibele structureringsmogelijkheden, en laten expliciete specificatie van de data beperkingen toe. Actueel zijn er zijn minstens dertig verschillende modellen beschikbaar. De meest bekende zijn: *entity-relationship* model, *binary* model, *semantic data* model en *infological* model.

Het *entity-relationship* (E-R) data model is gebaseerd op een waarneming van de reële wereld bestaande uit een verzameling objecten, *entiteiten*, met daartussen een aantal *relaties*. Een entiteit is een object dat bestaat en onderscheidbaar is van andere objecten. Dit onderscheid wordt gerealiseerd door aan elke entiteit een verzameling *attributen* toe te kennen welke de entiteit beschrijft. Een relatie is een associatie tussen verscheidene entiteiten. De verzameling van alle entiteiten van hetzelfde type en relaties van hetzelfde type worden respektievelijk *entity set* en *relationship set* genoemd.
Het schema kan voorgesteld worden in een E-R diagram. Een voorbeeld is gegeven in figuur 3.

1.3.2 Record gebaseerde modellen

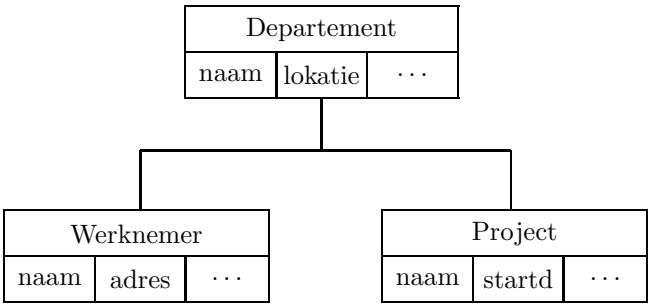
Deze modellen worden gebruikt om naast de globale logische structuur van de database ook een beschrijving te geven van de implementatie op een hoog niveau. Er is echter geen mogelijkheid om data beperkingen te specificeren. De drie meest verspreide data modellen zijn: de hiërarchische structuur, de netwerkstructuur en de relationele structuur.

Hiërarchische structuur : dit is een logische structuur waarbij elementen van de hiërarchie slechts ondergeschikt kunnen zijn aan één ander element. Het element aan de top noemt men de root.



Figuur 4: Het hiërarchisch model

Er kunnen dus enkel zuiver hiërarchische verbanden tussen entiteiten opgeslagen worden. Men kan dus vlot aangeven dat een bedrijf (figuur 4) bestaat uit departementen en elk departement uit afdelingen, en dat in elke afdeling een aantal werknemers zijn die elk een aantal kinderen hebben. Maar van zodra enige van die kinderen gemeenschappelijk zijn aan twee werknemers is de zuiver hiërarchische structuur verstoord en moet men die kinderen twee maal registreren of een andere kunstgreep uithalen.
In een hiërarchische databank kan een entiteit meerdere ondergeschikte entiteiten hebben. Dit betekent dat men in een hiërarchische databank kan opnemen dat elke werknemer behalve een aantal kinderen ook een aantal diploma's heeft. Omwille van deze hiërarchische structuur is de *kontrole* gemakkelijk maar is het geheel weinig *flexibel*.

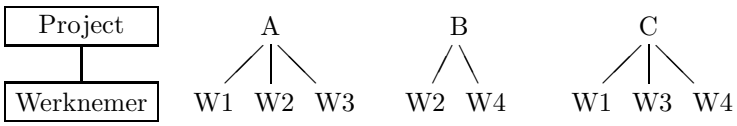


Een *Record* is een verzameling *velden*. Records van hetzelfde type worden gegroepeerd in *Record-types*.
Een *Parent-Child relationship* type (PCR-type) is een $1 : N$ relatie tussen twee Record types: langs de ene kant: een *parent* record type en langs de andere kant een *child* record type.

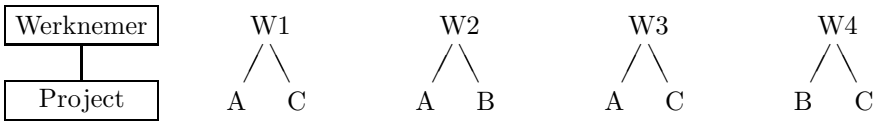
Een *hiërarchisch database schema* bevat een aantal hiërarchische schema's. Een hiërarchisch schema bevat een aantal Record-types en PCR-types en heeft volgende eigenschappen:

- Er is één Record-type dat in geen enkel PCR-type een child is: dit is de *root*.
- Elk Record-type, behalve de root, is steeds een child in juist één PCR-type.
- Een Record-type kan parent zijn in nul of meerdere PCR types.

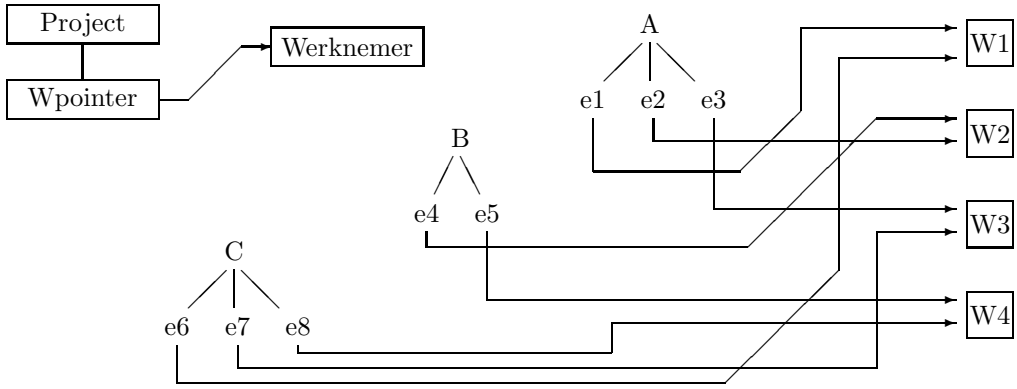
Een eerste probleem in dit model is dat $M : N$ relaties niet zo maar kunnen voorgesteld worden. Bijvoorbeeld, bij een project zijn verschillende werknemers betrokken en één werknemer kan in verschillende projecten ingeschakeld worden.
Dit wordt opgelost door records te dupliceren:



of andersom:

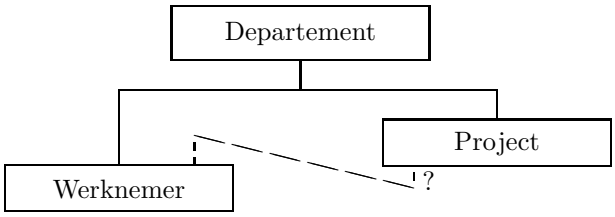


Er is ook een meer gesofistikeerde oplossing.
Een *virtueel record type* (VC) (of pointer) is een Record-type waarvan elke instantie een pointer bevat naar een record van een ander type (VP). Op die manier wordt een *virtuele parent-child relatie* voorgesteld tussen een het virtueel child (VC) en de virtuele parent (VP).
Met deze virtuele parent-child relaties kunnen $M : N$ relaties voorgesteld worden als $1 : N$ relaties:

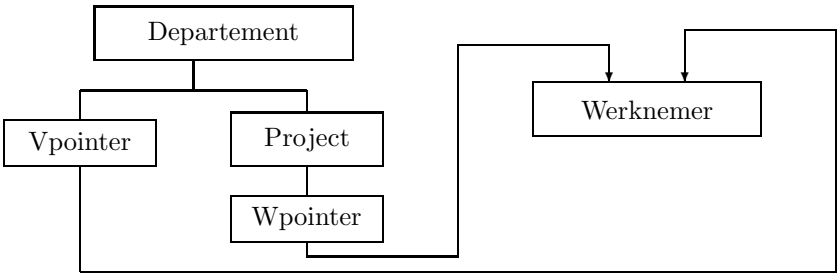


De relatie tussen `Werknemer` en `Wpointer` is een $1 : N$ relatie en is dus van het PCR type. Zo'n relatie wordt een *virtuele parent-child relatie* (VPCR type) genoemd. `Werknemer` is de *virtuele parent* en `Wpointer` is de *virtuele child*. Conceptueel zijn PCR en VPCR types hetzelfde. Het verschil ligt bij de implementatie. Een PCR type wordt gewoonlijk geïmplementeerd als een hiërarchische sequentie. Bij een VPCR type wordt gebruik gemaakt van een pointer van de virtuele kind-record naar de virtuele parent.

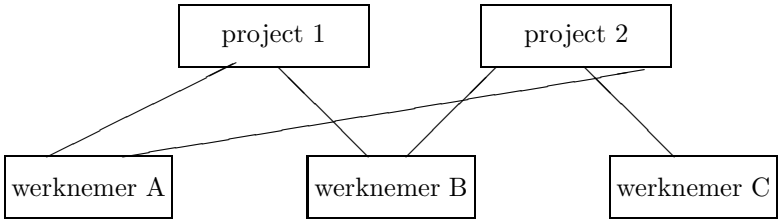
Een tweede probleem treedt op wanneer een Record-type een child is van meer dan één PCR-type. Bijvoorbeeld een werknemer die lid is van een departement en ook aan een project werkt.
Dit kan weer opgelost worden door zeer veel records te dupliceren.



Het probleem dat een Record-type child is in meer dan één PCR type, kan ook opgelost worden met VPCR types:



Netwerkstructuur : in een dergelijke structuur zou elk element in relatie kunnen staan met elk ander element. Deze structuur is natuurlijk veel flexibeler dan de hiërarchische structuur, maar biedt veel moeilijker kontrole.

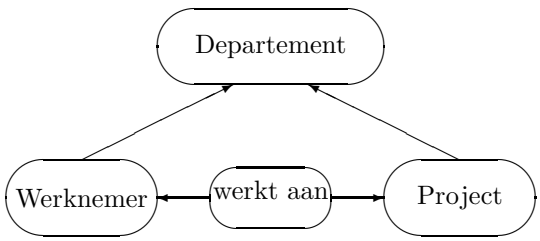


Figuur 5: Het netwerk model

Voorbeeld: Gegevens over een aantal projecten en een aantal werknemers (figuur 5): elk projekt kan in principe elke werknemer gebruiken, en elke werknemer kan in principe bij elk projekt betrokken zijn.
De fysische opslag van de gegevens gebeurt natuurlijk niet volgens de logische structuur (bijv. in niveaus) maar gewoon lineair. De logische structuur wordt opgebouwd met behulp van *links* (verwijsadressen die in een apart veld van het gegevenselement worden geplaatst). De structuur van een netwerk databank wordt hierdoor vrij complex, en een probleem is dat men alle mogelijke verbanden tussen entiteiten op voorhand (dus bij het ontwerp) moet specificeren.
Het netwerk model kan gezien worden als een ER-model, beperkt tot binaire veel-op-één relaties. Er zijn twee belangrijke elementen:

1. logisch record type: te vergelijken met de entity-verzameling, bevat dus een aantal records;
2. link: een veel-op-één binaire relatie: een connectie tussen twee logische record types, namelijk tussen het *lid* type naar het *eigenaar* type.

De voorstelling gebeurt met ovalen en pijlen:



Vier logische record types.
Vier links:

- Departement (eigenaar van lid) Werknemer
- Departement (eigenaar van lid) Project
- Werknemer (eigenaar van lid) werkt aan
- Project (eigenaar van lid) werkt aan

Relationele structuur : in een relationele databank zijn geen expliciete links aanwezig, en alle informatie (entiteiten en relaties ertussen) wordt uniform voorgesteld in tabellen die men *relaties* noemt.
Voordelen:

wnr	naam	adres
124	Appels	Genk
167	Aerts	Gent
482	Bols	Geel
512	Dams	Bree

wnr	pro
124	24
167	135
167	739
482	135
512	739

pro	naam	startd
24	karton	15/11/2001
135	pennen	07/02/2002
739	dozen	23/01/2002

Tabel 1: Het relationele model

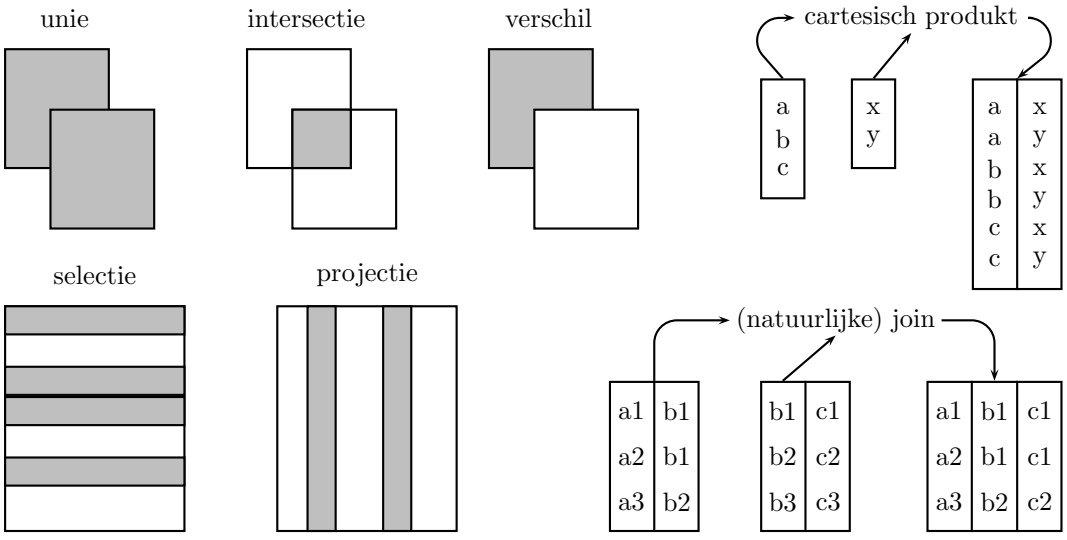
1. zeer eenvoudig model, gegrondvest op een stevige mathematische basis;
2. files waarin de gegevens opgeslagen worden hebben een eenvoudige structuur, verwant met klassiek geïndiceerde bestanden;
3. eindgebruiker kan zich deze relaties gemakkelijk voorstellen als gewone tweedimensionale tabellen;
4. op deze relaties (tabellen) zijn een aantal bewerkingen mogelijk (bijv. selectie, projectie, join), die de basis vormen van een relationele taal. Zo'n bewerking gebeurt op de relatie als geheel i.p.v. record per record verwerking:

   selectie: bepaling van een aantal tuples die aan een voorwaarde voldoen;

   projectie: maken van een relatie die uit een deelverzameling attributen van een gegeven relatie bestaat;

   join: creatie van een nieuwe relatie met tuples uit twee oorspronkelijke relaties.

   Ook uit de verzamelingenleer zijn een aantal bewerkingen overgenomen.



Figuur 6: Traditionele set operatoren en speciale relationele operatoren

Het belangrijkste nadeel dat nu nog aan relationele databanken kleeft is de lagere efficiëntie, maar betere algoritmes (o.a. zoekstrategieën), snellere machines en gespecialiseerde hardware zullen dit in de toekomst zeker verhelpen.

1.4 Data definitie taal en data manipulatie taal

Een database schema wordt gespecificeerd door een verzameling definities neergeschreven in een speciale taal, de *data definition language* (DDL). Het resultaat van de compilatie van DDL sta-

tements is een verzameling tabellen welke in een speciale tabel gestockeerd worden, namelijk de *data dictionary* of *directory*.
Een data dictionary is een bestand dat *metadata* bevat: "data over data". Dit bestand wordt geconsulteerd telkens echte data gelezen of aangepast wordt in het database systeem.
Een speciaal type van DDL is de *data storage and definition language*. Deze wordt gebruikt om geheugen structuur en de access metodes te specificeren. De implementatie details van de database schema's worden hiermee verborgen gehouden voor de gewone gebruiker.

Data manipulatie omvat:

- opvragen van informatie
- toevoegen van nieuwe informatie
- verwijderen van informatie

Een *data manipulation language* (DML) is een taal die de gebruikers in staat stelt toegang te hebben tot de data. Er zijn twee types: **procedureel** waarbij de gebruiker specificeert *welke* data hij nodig heeft en *hoe* hij deze kan vinden en **niet-procedureel** waarbij de gebruiker alleen moet specificeren *welke* data hij nodig heeft.
Niet-procedurele talen zijn gewoonlijk gemakkelijker aan te leren en te gebruiken dan procedurele. Maar ze kunnen code genereren die niet erg efficiënt is zodat optimisatie technieken nodig zijn.
Een *query* is een statement om informatie op te vragen. Dat deel van een DML dat betrekking heeft tot informatie opvraging, wordt een *query language* (vraagtaal) genoemd. Alhoewel het technisch niet juist is, worden in praktijk vraagtaal en data manipulatie taal als synoniemen gebruikt.

1.5 Database administrator

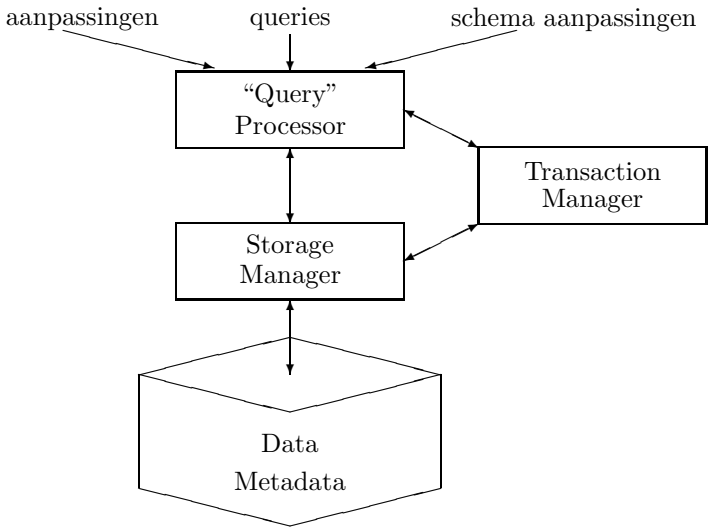
Eén van de belangrijkste redenen voor het hebben van een DBMS is de centrale controle over gegevens en de programma's die deze gegevens bewerken. De persoon die de centrale controle over het systeem heeft, wordt de *database administrator* (DBA) genoemd. Zijn functies omvatten:

- Definitie van het schema: de creatie van het eerste database schema: het schrijven van een verzameling definities welke door de DDL compiler vertaald worden in een verzameling tabellen welke permanent in de *data dictionary* gestockeerd worden.
- Definitie van de geheugen structuur en access methodes: een verzameling definities omtrent de fysische organisatie.
- Aanpassingen aan het schema en de fysische organisatie.
- Toegang verlenen aan de verschillende gebruikers: aangeven welke delen van de database kunnen gebruikt worden door welke gebruikers. Niet elke gebruiker heeft behoefte aan of heeft toelating tot alle gegevens in de database en daarom wordt hem eventueel slechts beperkte toegang gegeven.
- Specifikatie van de integriteitsbeperkingen: deze worden in een speciale systeemstructuur bewaard en geraadpleegd telkens er een update van een gegeven gedaan wordt. De data in de database moet voldoen aan bepaalde types van consistentie beperkingen. De beperkingen moeten gecontroleerd worden telkens er een aanpassing aan de data gebeurt; indien er niet aan voldaan is moet een aangepaste actie uitgevoerd worden.
- Ontwikkelen van backup procedures: om de gegevens te kunnen herstellen na een faling van het systeem.

1.6 De architectuur van een DBMS

In figuur 7 worden de verschillende onderdelen van een DBMS getoond. Onderaan is de plaats voorgesteld waar de data gestockeerd wordt; gewoonlijk is dit één of meerdere disks. Deze component bevat niet alleen gewone, echte data maar ook *metadata*. Dit is informatie over de structuur van de data. Bij een R-DBMS bijvoorbeeld bevat de metadata de namen van de relaties, de namen

van de attributen van deze relaties en de datatypes van deze attributen (integer, string, ...). Een DBMS bevat normaal ook *indexen* voor de data. Een index is een datastructuur die het zoeken van informatie in de databank versnelt.



Figuur 7: Belangrijkste componenten van een DBMS

Storage manager. Zijn taak bevat het ophalen van de gevraagde data uit de databank en het aanpassen van de informatie op aanvraag van de bovenliggende niveaus. In een eenvoudig DBMS is deze component gewoon het filesysteem van het onderliggende besturingssysteem. De naakte data wordt op disk gestockeerd waarbij het filesysteem gebruikt wordt dat normaal deel uitmaakt van het besturingssysteem. De storage manager vertaalt de verschillende DML statements in low-level filesysteem commando's en is dus verantwoordelijk voor de daadwerkelijke stockage, opvragen en aanpassen van de data in de databank. Omwille van de efficiëntie beheert een DBMS meestal zelf de data op de disk. Er zijn twee onderdelen:

file manager : beheert de locatie van de bestanden op de disk; levert het blok of de blokken van een bestand op aanvraag van de buffer manager;

buffer manager : stockeert het door de file manager geleverde blok in een pagina van het primair geheugen; dit blok blijft gedurende een bepaalde tijd in primair geheugen zodat andere queries deze data ook kunnen gebruiken zonder dat er van disk gelezen moet worden; na een tijd, wanneer er geen aanvragen voor dat blok meer blijken te zijn, wordt de pagina voor een ander net ingelezen blok gebruikt.

Query processor. Deze component doet meer dan queries afhandelen. Ook de vragen voor aanpassingen van de data en de metadata passeren via de query processor. Deze vragen worden meestal uitgedrukt in een taal van *hoog* niveau (bijv. SQL). De query processor vertaalt de vraag naar een reeks bevelen die naar de storage manager gestuurd worden, die ze dan zal uitvoeren.
Het moeilijkste deel is de *query optimisatie*: de keuze van een goede opeenvolging van data-aanvragen aan het storage systeem zodat snel de gevraagde data gevonden wordt. Hiervoor worden indexen gebruikt, maar ook de volgorde waarin de verschillende stappen van een complexe query uitgevoerd worden is meestal bepalend voor de snelheid.

Transaction manager. Deze component is verantwoordelijk voor de integriteit van het systeem. Hij moet verzekeren dat verschillende queries die simultaan lopen niet met elkaar interfereren.

Concurrentie controle: wanneer verschillende gebruikers de database gelijktijdig aanpassen, is de consistenstie van de data misschien niet meer gegarandeerd. Het is noodzakelijk voor het systeem om de interactie tussen de verschillende gelijktijdige gebruikers te controleren. Het systeem mag ook geen data verliezen, zelfs bij een systeemcrash.
Via de interactie met de query processor komt de transaction manager te weten op welke data de actuele queries operaties uitvoeren zodat conflicterende acties kunnen vermeden worden. Het is mogelijk om bepaalde queries of operaties uit te stellen zodat er geen conflicten optreden.
Er is ook interactie met de storage manager: voor de bescherming van de data moet er gewoonlijk een *log* bijgehouden worden van de veranderingen op de data. Bij een goede ordening van de operaties zal de log een lijst van een aanpassingen bevatten die na een systeemcrash terug kunnen uitgevoerd worden.

Invoertypes. Men kan vier types van gebruikers onderscheiden: *naïeve gebruikers* via applicatie interfaces, *applicatie programmeurs* via applicatieprogramma's, *gesophisticeerde gebruikers* via queries en *database adminstrators* die zich bezig houden met het schema van de databank.

Queries : vragen naar informatie. Zo'n vraag kan op twee manieren gegenereerd worden. Via een *generisch query interface* kunnen SQL statements ingetikt worden. Deze worden doorgegeven aan de query processor die een antwoord teruggeeft. Een andere manier zijn de *application program interfaces*. In een gebruiksvriendelijk programma (met GUI) kan de gebruiker aangeven welke gegevens gewenst zijn; het programma zet deze vraag zelf om in SQL statements die door de query processor uitgevoerd worden. Het resulaat wordt zo elegant mogelijk aan de gebruiker gepresenteerd.

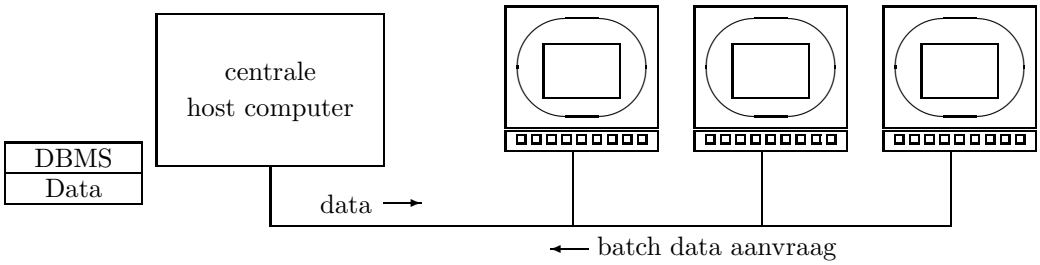
Aanpassingen : operaties om de gegevens te veranderen; eventueel zijn dit toevoegingen of worden er gegevens verwijderd. De manier waarop is zoals bij queries.

Schema aanpassingen : commando's die gewoonlijk gegeven worden door geauthoriseerd personeel, bijvoorbeeld de *database administrator*, die de toelating hebben om het schema aan te passen of een nieuwe databank te creëren.

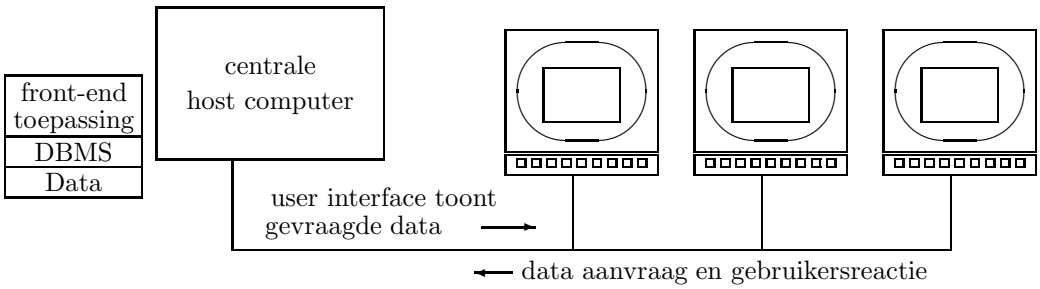
1.7 Client-server architectuur

In een client-server architectuur worden aanvragen door één proces (de client) verzonden naar een ander proces (de server) om daar uitgevoerd te worden. In een databanktoepassing is het volledige DBMS een server, behalve de query interfaces die interageren met de gebruiker. De client stelt een vraag mbv. SQL naar de server. De database server antwoordt in de vorm van een tabel of een relatie. Er is wel een trend om meer werk door de client te laten doen omwille van het ontstaan van een bottleneck in de server wanneer er zeer vele simultane databankgebruikers zijn.

Historisch overzicht. De eerste database toepassingen draaiden op grote centrale computers via *domme* terminals en later intelligente terminals of workstations. Omdat alles vrij duur was, werd de interactie met de computer beperkt via *batch* data aanvragen naar de centrale computer (figuur 8).
Data opvragen en display werd op terminals gedaan. Deze configuratie wordt nog steeds veel gebruikt. Er is wel een evolutie zodat de applicaties op de centrale computer een betere gebruikersinterface kregen. Display gebeurt nog steeds op een terminal maar de verwerking van de gebruikersinteractie wordt uitgevoerd door de centrale computer (figuur 9).
Dit vereiste meer computerkracht omdat de computer nu niet alleen de aanvraag voor gegevensverwerking moet behandelen maar ook de interacties van elke individuele gebruiker.
Met de introductie van PCs met voldoende lokale verwerking en stockage mogelijkheden, werden programma's zoals dBase en Lotus enorm populair (figuur 10). Gebruikers konden nu zelf hun eigen lokale data bewerken afzonderlijk van de gegevens gestockeerd in de grote centrale computer. Deze PCs boden ook een meer grafische userinterface (GUI) die gemakkelijker om te gebruiken was en ook interactiever. Echter, elke nieuwe PC applicatie stockeerde de data op zijn eigen



Figuur 8: Verwerking gebaseerd op terminals

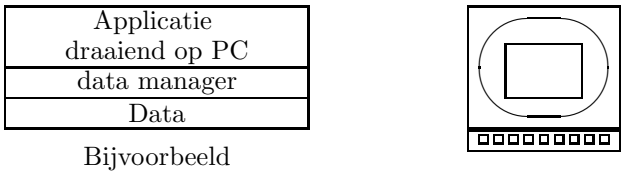


Figuur 9: Gebruikersinterface op host

manier zodat snel data op de meest verschillende plaatsen en in de meest verschillende formaten gestockeerd werd.
Volgende stap was de introductie van een LAN. Gebruikers gingen hun PCs met elkaar verbinden, waarbij ook een file-server voorzien werd om gemeenschappelijke data te stockeren (figuur 11). De file-server computer had als taak de gegevens te bewaren en volledige bestanden naar PCs door te zenden wanneer deze er om vroegen. De PC kreeg zo meer tijd om de data lokaal te verwerken.
Deze methode werkte goed zolang het aantal gebruikers en de hoeveelheid data dat op de file-server aanwezig is, beperkt bleef. De file-server werd echter snel een bottleneck bij het bewaren van grote hoeveelheden data of wanneer meer en meer gebruikers de centraal gestockeerde data begonnen op te vragen. Er ontstond zo ook een verhoogde trafiek op het netwerk. Daarenboven was de file server niet voldoende in staat om een aantal bijkomende taken te vervullen: beveiliging en onderhoud van de integriteit van de data, afhandelen van concurrente updates door verschillende gebruikers, backup en herstel procedures.

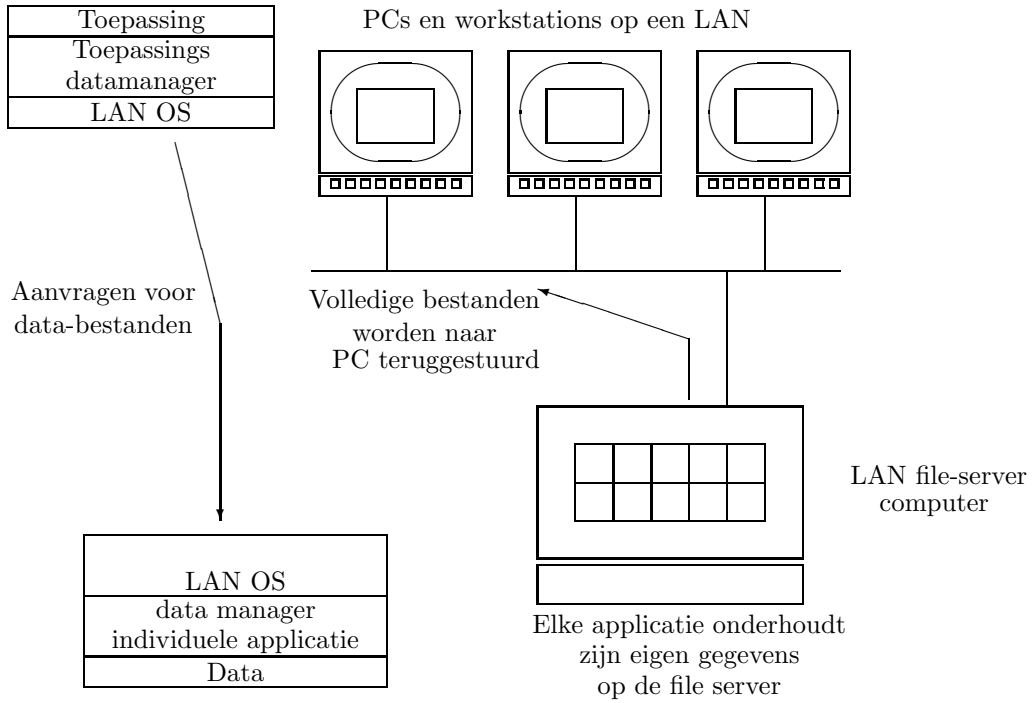
Client-database-server architectuur. De tekorten van de file-server technologie hebben tot de ontwikkeling van producten geleid die de C/S architectuur gebruiken. Deze configuraties proberen op de beste manier gebruik te maken van zowel hardware als software hulpmiddelen door de functies op te delen in twee:

- het front-end gedeelte van de toepassing dat uitgevoerd wordt op client computers of work-



Bijvoorbeeld
spreadsheet, database,
grafieken, presentaties, ... die draaien op PC hardware

Figuur 10: Stand-alone applicaties



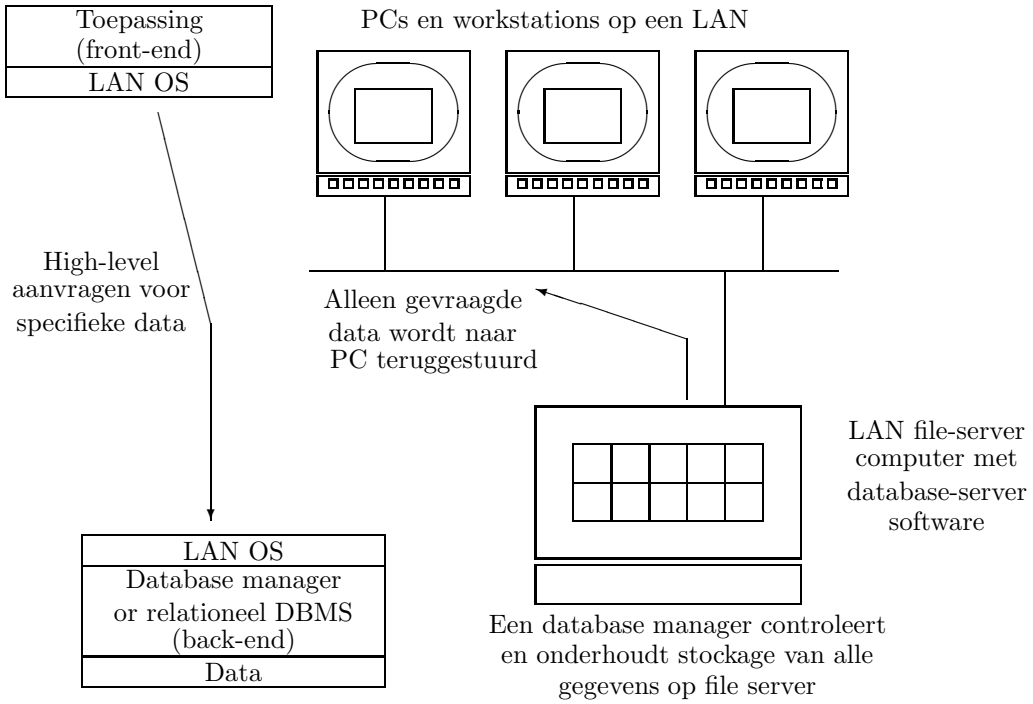
Figuur 11: File Server architectuur

stations;

- de back-end database server, welke de data stockeert en aanvragen afhandelt.

Figuur 12 illustreert deze architectuur met een database server. Data op de database server wordt slechts eenmaal gestockeerd en kan tegelijk (concurrent) opgevraagd worden door vele verschillende applicaties, o.a. databases, spreadsheets en tekstverwerkers. De database server verwerkt de data-aanvragen en stuurt alleen de gevraagde data terug naar de applicaties op de client PCs. De PC is alleen verantwoordelijk voor de applicatie van de gebruiker: de afhandeling van de interactie met de gebruiker en het genereren van data-aanvragen. In plaats van het verwerken van de data, kan de client PC zich focusseren op de gebruikersapplicatie met behulp van steeds meer gesofistikeerde GUIs beschikbaar op PC of workstations. De database server houdt zich alleen met database beheer bezig en kan dus zorgen voor het onderhoud van de gegevens-integriteit, foutafhandeling en beveiligingscontrole. Daarenboven wordt ook de mogelijkheid voor de gebruiker geboden om concurrent toegang tot gegevens te hebben en deze ook aan te passen.
Voordelen van client-server verwerking:

- Een efficiëntere verdeling van het werk. Zowel de client als de server krijgen taken toegewezen waarvoor ze het beste geschikt zijn. De client computer neemt de presentatie van een grafische user interface voor zich, o.a. het afhandelen van de interactie tussen gebruiker en toepassing. De database server houdt zich onledig met de verwerking van grote volumes data op een hoog-performante manier met controles voor beveiliging, integriteit en concurrency.
- Mogelijkheden voor zowel horizontale als vertikale schaling van de resources om de taken uit te voeren. Horizontaal, door de dataverwerkingsjobs (opvragen en updates) te verdelen over de verschillende processoren op het netwerk. Vertikaal door het RDBMS te verhuizen naar een grotere, krachtigere computer.
- Toepassingen op basis van de C/S architectuur kunnen gemakkelijker op een kleinere client computer uitgevoerd worden met een betere performantie. Omdat het merendeel van het



Figuur 12: Client-Server architectuur

database werk *offloaded* is naar de server, kan een goedkopere PC gebruikt worden voor de applicatie zelf. Ook de trafiek op het netwerk is gereduceerd omdat de applicaties alleen specifieke data aanvragen naar de server sturen en omdat alleen de gevraagde data door de server naar de client teruggestuurd wordt.

- Gebruikers kunnen hun vertrouwde en favoriete tools op PC blijven gebruiken. Een groot deel van de bestaande applicaties zijn reeds aangepast zodat ze data op servers kunnen opvragen. Nieuwe applicaties worden zodanig geschreven dat ze kunnen gebruik maken van de C/S configuraties. Omdat betere en eenvoudigere tools het ontwikkelen van toepassingen gemakkelijker maken, kan het voorkomen dat de eindgebruiker zijn eigen toepassing zelf ontwerpt waardoor de ontwikkelingstijd gereduceerd wordt.
- Clients hebben toegang tot meer data. Door de standaard SQL die op heel wat servers gebruikt wordt, kan men toegang tot data krijgen op een grote verscheidenheid van machines en wordt het overdragen van de applicatie naar een ander platform gemakkelijker.
- Belangrijke, waardevolle gegevens kunnen op de juiste manier beveiligd worden tegen verlies of niet toegelaten gebruik. Dataverwerking wordt uitgevoerd op het centrale DBMS, die hiervoor specifiek uitgerust is. Belangrijke aspecten van database toepassingen zoals beveiliging, gegevensintegriteit, concurrency. backup en recovery worden terug door gespecialiseerde informatici uitgevoerd.
- Goedkopere en krachtigere PC hardware en software resulteren in oplossingen die gemakkelijker te implementeren zijn dan de klassieke database toepassingen.

2 Analyse van gegevens: entity-relationship model

2.1 Entiteiten en entiteit-verzamelingen

Een *entiteit* is een object dat bestaat en onderscheidbaar is van andere objecten. Bijvoorbeeld Jan Peeters met studentnummer 89204 is een entiteit omdat het op een unieke manier een specifieke persoon in het universum identificeert. Een entiteit kan concreet zijn, zoals een persoon of een boek, of abstract zoals een vakantiedag of een concept.
Een *entiteit-verzameling* is een verzameling van entiteiten van hetzelfde type. De verzameling van alle personen die aan een bepaald instituut studeren, kan gedefinieerd worden als de entiteit-verzameling *student*. Entiteit-verzamelingen moeten niet disjunct zijn. Het is bijvoorbeeld mogelijk de entiteit-verzameling *docent* en de entiteit-verzameling *student* van een bepaald instituut te definiëren. Een *persoon* entiteit kan een *student* entiteit of een *docent* entiteit of beiden zijn.
Een entiteit wordt voorgesteld door een verzameling *attributen*. Mogelijke attributen voor de *student* entiteit zijn *snaam*, *studnr*, *straat* en *woonplaats*. Voor elk attribuut bestaat er een verzameling van toegelaten waarden, het *domein* van dat attribuut. Het domein van het attribuut *naam* kan bijvoorbeeld de verzameling van alle tekst strings met een bepaalde lengte zijn.
Formeel is een attribuut een functie die een entiteit-verzameling afbeeldt op een domein. Dus elke entiteit wordt beschreven door een verzameling van (attribuut, waarde) paren, een paar voor elk attribuut van de entiteit-verzameling.
In de volgende voorbeelden zullen volgende entiteit-verzamelingen gebruikt worden:

- **student** met attributen *snaam*, *studnr*, *straat* en *woonplaats*;
- **biografie** met attributen *geboortepl* en *geboortedat*;
- **docent** met attributen *dnaam*, *docnr* en *acadgr*;
- **vak** met attributen *vnaam*, *uren* en *vaknr*;
- **richting** met attributen *fase*, *opleiding* en *minor*;
- **uitslag** met attributen *percentage* en *vermelding*.

Een databank is een collectie van entiteit-verzamelingen, welke elk een aantal entiteiten van hetzelfde type bevatten.

2.2 Attributen

Sommige attributen kunnen verdeeld worden in kleinere delen met een eigen betekenis. Een *adres* attribuut bijvoorbeeld kan onderverdeeld worden in een *straatadres*, *postcode* en *woonplaats*. Een attribuut dat is samengesteld uit een aantal attributen wordt **samengesteld** genoemd, terwijl attributen die ondeelbaar zijn **eenvoudig** of **atomisch** genoemd worden. Samengestelde attributen kunnen een hiërarchie vormen; *straatadres* bijvoorbeeld kan verder onderverdeeld worden in *straatnaam*, *nummer* en *busnr*. Samengestelde attributen zijn nuttig wanneer een gebruiker soms het samengestelde attribuut als een eenheid wil beschouwen en op andere momenten specifiek de componenten wil refereren.
De meeste attributen hebben één enkelvoudige waarde voor een specifieke entiteit; zij worden **single-valued** genoemd. De entiteit Student bijvoorbeeld heeft één waarde voor het attribuut *leeftijd*. In sommige gevallen kan een attribuut een verzameling van waarden hebben voor een specifieke entiteit. Het attribuut *academische graad* kan voor sommige personen leeg zijn, andere personen hebben één academische graad, terwijl er ook personen zijn met twee of meer academische graden. Zo'n attributen worden **multi-valued** genoemd. Een multi-valued attribuut kan een beneden- en bovengrens hebben op het aantal waarden voor een individuele entiteit.
In sommige gevallen kunnen twee (of meer) attributen met elkaar gerelateerd zijn, bijvoorbeeld leeftijd en geboortedatum van een persoon. Voor een specifieke persoon kan de waarde van leeftijd bepaald worden op basis van de huidige datum en de waarde van geboortedatum. Het leeftijd

attribuut wordt het **afgeleide** attribuut genoemd en is dus **afleidbaar** van het geboortedatum attribuut. Sommige attribuut waarden kunnen afgeleid worden van *gerelateerde entiteiten*; bijvoorbeeld het *aantal werknemers* attribuut van een departement entiteit kan berekend worden door het aantal werknemers in dat departement te tellen.
Soms heeft een specifieke entiteit geen realistische waarde voor een attribuut, bijvoorbeeld het *busnr* attribuut in een adres. In andere gevallen kan het zijn dat het attribuut wel betekenis heeft voor de entiteit maar dat de waarde niet gekend is. Voor zo'n situaties is de speciale waarde **null** gecreëerd. Deze waarde kan twee betekenissen hebben: *niet van toepassing* en *ongekend*.

2.3 Relaties en relatie-verzamelingen

Een *relatie* is een associatie tussen verschillende entiteiten. Men kan bijvoorbeeld een relatie definiëren welke "Jan Peeters" associeert met richting "3cbio". Deze relatie specificeert dat Jan Peeters een student is die in het derde jaar zit van de opleiding chemie en daarin de minor biochemie volgt.
Een *relatie-verzameling* is een verzameling van relaties van hetzelfde type. Formeel is het een wiskundige relatie op $n \geq 2$ entiteit-verzamelingen. Indien $E_1, E_2, \ldots, E_n$ entiteit-verzamelingen zijn, dan is de relatie-verzameling R een deelverzameling van

$$\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

waarbij $(e_1, e_2, \ldots, e_n)$ een relatie is.
Tussen de twee entiteit-verzamelingen *student* en *richting* kan men de relatie-verzameling *StRi* definiëren welke een associatie tussen studenten en richtingen voorstelt. Deze relatie (*StRi*) is een voorbeeld van een binaire relatie-verzameling, er zijn namelijk twee entiteit-verzamelingen bij betrokken. Soms gebruikt men relatie-verzamelingen waarbij meer dan twee entiteit-verzamelingen bij betrokken zijn. De relatie *SRU* is gedefinieerd tussen drie verzamelingen en geeft weer dat een *student* in een bepaalde *richting* een specifieke *uitslag* behaald heeft.
De functie die een entiteit vervult in de relatie wordt *rol* genoemd. Normaal zijn rollen impliciet en worden gewoonlijk niet gespecificeerd. Ze zijn nochtans nuttig wanneer de betekenis van een relatie moet verduidelijkt worden. Dit is het geval wanneer de entiteit-verzamelingen van een relatie-verzameling niet verschillend zijn. In de relatie-verzameling *werkt-voor* tussen geordende paren van de *docent* entiteit kan het eerste element van het geordende paar de rol van manager hebben en het tweede de rol van ondergeschikte.
Een relatie kan ook beschrijvende attributen hebben. Zo kan *bis* een attribuut zijn van de *StRi* relatie-verzameling. Dit attribuut specificeert of de student deze richting voor de eerste of de tweede maal volgt.

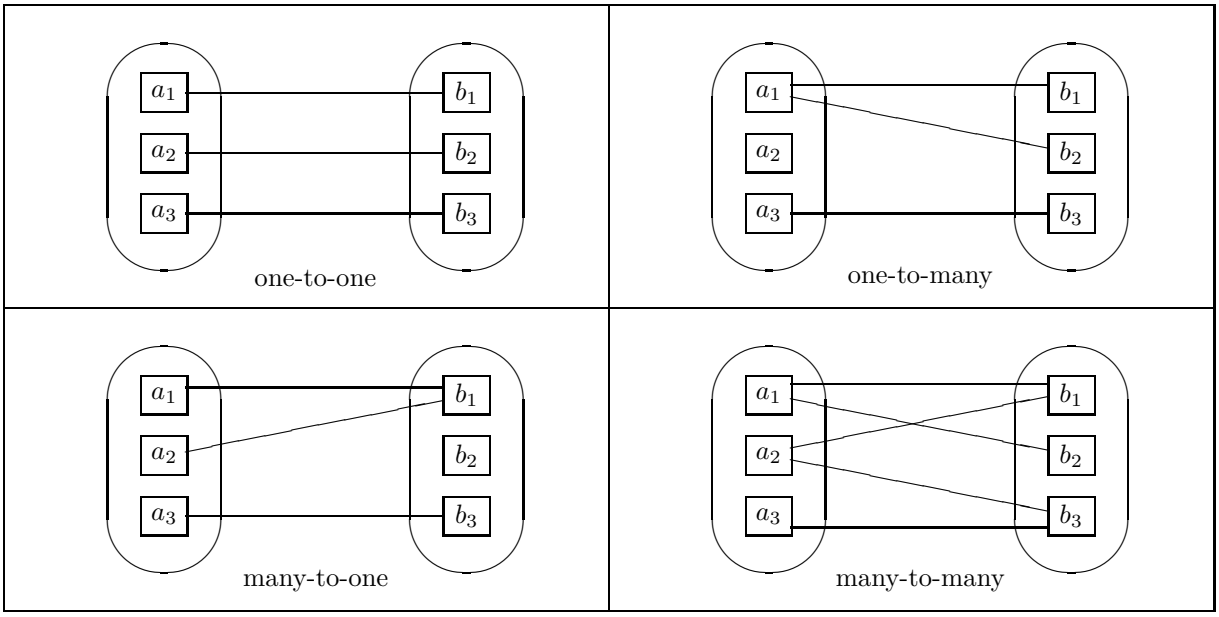
2.4 Integriteitsbeperkingen

In het globale E-R schema kunnen bepaalde beperkingen gedefinieerd worden, waaraan de inhoud van de databank moet voldoen. De *bestaansbeperking* is een beperking op het domein van waarden dat een bepaald attribuut kan aannemen. Bijvoorbeeld moet de geboortedatum van een student gelegen zijn na 1940.
Een belangrijke beperking is de *mapping cardinaliteit* welke het aantal entiteiten weergeeft dat met een andere entiteit kan geassocieerd worden via een relatie-verzameling.
Voor een binaire relatie-verzameling R tussen entiteit-verzamelingen A en B is de mapping cardinaliteit één van de volgende.

- **Eén-op-één**: een entiteit in A is geassocieerd met ten hoogste één entiteit in B, en een entiteit in B is geassocieerd met ten hoogste één entiteit in A.
- **Eén-op-veel**: een entiteit in A is geassocieerd met een willekeurig aantal entiteiten in B, maar een entiteit in B is geassocieerd met ten hoogste één entiteit in A.

- **Veel-op-één**: een entiteit in A is geassocieerd met ten hoogste één entiteit in B, maar een entiteit in B kan met een willekeurig aantal entiteiten in A geassocieerd zijn.
- **Veel-op-veel**: een entiteit in A is geassocieerd met een willekeurig aantal entiteiten in B, en een entiteit in B kan met een willekeurig aantal entiteiten in A geassocieerd zijn.

Deze verschillende mapping cardinaliteiten zijn voorgesteld in figuur 13.



Figuur 13: De verschillende mapping cardinaliteiten

De juiste mapping cardinaliteit voor een specifieke relatie-verzameling is natuurlijk afhankelijk van de reële wereld welke men wil modelleren met de relatie-verzameling. Afhankelijk van instituut tot instituut kan een student slechts één of meerdere richtingen volgen. In de eerste geval is de relatie-verzameling *StRi* **veel-op-één**, in het tweede geval heeft men een **veel-op-veel** associatie. Een andere soort beperking is de *bestaans-afhankelijkheid*. Wanneer het bestaan van een entiteit x afhankelijk is van het bestaan van de entiteit y, dan is *x bestaans-afhankelijk van y*. Praktisch betekent dit dat wanneer y verwijderd wordt, ook x verdwenen is. Entiteit y is de *dominante entiteit* en x is de *ondergeschikte entiteit*.
Tussen de entiteit-verzamelingen *richting* en *vak* kan de relatie *RiVak* gedefinieerd worden. Deze specificeert dat in een bepaalde richting verschillende vakken gedoceerd worden. Het is een **één-op-veel** relatie. Elke vak-entiteit moet met een richting geassocieerd zijn. Als de richting-entiteit verwijderd wordt, dan moeten alle ermee geassocieerde vakken verwijderd worden. Daarentegen kunnen vakken verwijderd worden zonder effect op de richting-entiteit. De entiteit-verzameling *richting* is **dominant** en *vak* is **ondergeschikt** in de RiVak relatie.

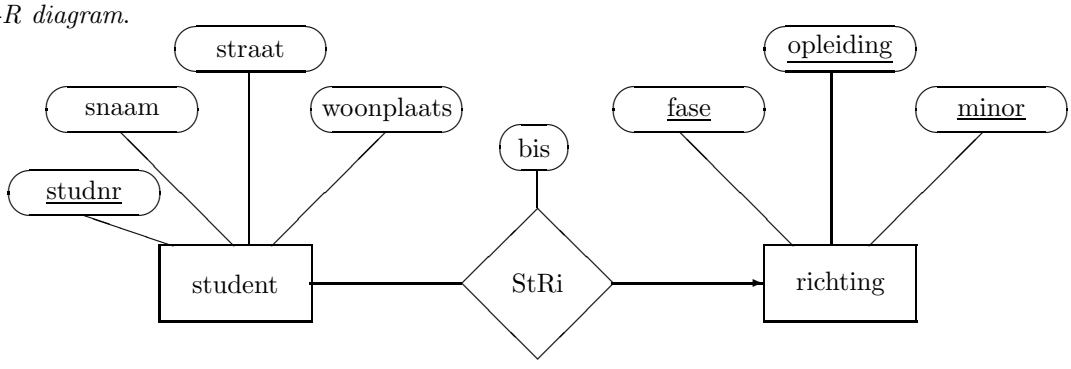
2.5 Primaire sleutels

Een belangrijke taak bij het opstellen van het database model is aangeven hoe entiteiten en relaties onderscheiden worden. Conceptueel zijn individuele entiteiten en relaties verschillend maar voor een database moeten deze verschillen uitgedrukt worden in termen van attributen. Om zo'n onderscheid te maken wordt aan elke entiteit-verzameling een *supersleutel* toegekend. Een supersleutel is een verzameling van één of meerdere attributen welke tesamen de gebruiker toelaten een entiteit uniek te identificeren in een entiteit-verzameling. Het *studnr* attribuut van de entiteit-verzameling *student* is bijvoorbeeld voldoende om één student van een andere te onderscheiden.

Dus *studnr* is een supersleutel. Ook kan de combinatie *snaam* en *studnr* als supersleutel voor de entiteit-verzameling *student* genomen worden. Het *snaam* attribuut van de entiteit-verzameling *student* is geen supersleutel omdat verschillende mensen dezelfde naam kunnen hebben.
Wanneer K een supersleutel is, is ook elke superverzameling van K een supersleutel. Meestal is men echter geïnteresseerd in de kleinst mogelijke supersleutel, d.i. een supersleutel waarvan geen enkele eigenlijke deelverzameling ook een supersleutel is. Zulke minimale supersleutels worden *kandidaatsleutel*s genoemd.
Het is mogelijk dat verschillende verzamelingen van attributen als kandidaatsleutel kunnen dienen. Een combinatie van *snaam* en *straat* kan bijvoorbeeld voldoende zijn om de verschillende elementen van de entiteit-verzameling *student* te onderscheiden. Dus zowel {studnr} als {snaam,straat} zijn kandidaatsleutels.
De term *primaire sleutel* wordt gebruikt om de kandidaatsleutel aan te duiden welke door de database ontwerper gekozen is als voornaamste middel om entiteiten in een entiteit-verzameling te identificeren.
Het is mogelijk dat een entiteit-verzameling niet voldoende attributen heeft om een primaire sleutel te vormen. Alhoewel elke *vak* entiteit onderscheidbaar is, kunnen vakken uit verschillende richtingen dezelfde *vaknr* hebben. Dus heeft deze entiteit-verzameling geen primaire sleutel. Zo'n entiteit-verzameling krijgt de naam *zwakke entiteit*. Een entiteit met een primaire sleutel wordt *sterke entiteit* genoemd.
Het concept van sterke en zwakke entiteiten is gerelateerd met het "bestaans afhankelijkheid" concept. Een sterke entiteit is per definitie een dominante entiteit, een zwakke entiteit is een ondergeschikte entiteit.
Een zwakke entiteit heeft geen primaire sleutel. Toch moet er een middel zijn om tussen al deze elementen van de entiteit-verzameling die entiteiten te onderscheiden die afhankelijk zijn van een bepaalde sterke entiteit. De *discriminator* van een zwakke entiteit-verzameling is de verzameling attributen die toelaat het onderscheid te maken. In het voorbeeld is *vaknr* de discriminator van de zwakke entiteit-verzameling.
De primaire sleutel van een zwakke entiteit-verzameling wordt gevormd door de primaire sleutel van de sterke entiteit-verzameling, waarvan ze bestaans-afhankelijk is, en haar eigen discriminator.
Relatie-verzamelingen hebben ook primaire sleutels. Ze worden gevormd door de attributen van de primaire sleutels van de entiteit-verzamelingen die de relatie-verzameling definiëren.

2.6 ER-diagram

De globale logische structuur van een databank kan grafisch voorgesteld worden door middel van een *E-R diagram*.



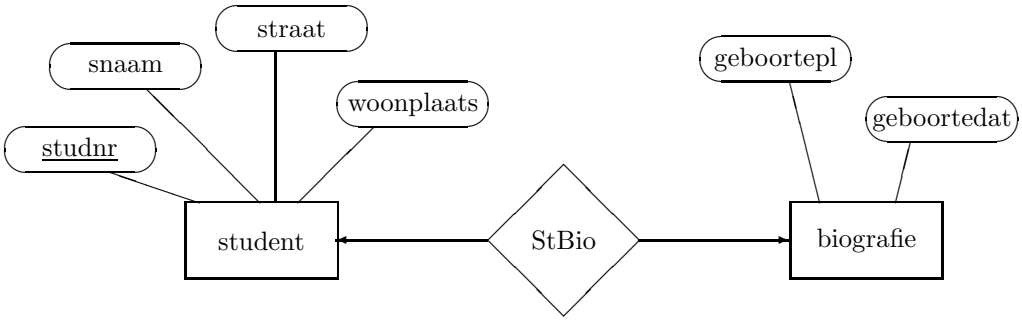
Figuur 14: Entity-Relationship diagram

Zo'n diagram bestaat uit de volgende componenten:

- **Rechthoeken**: voorstelling van entiteit-verzamelingen.
- **Ellipsen** (ovalen): voorstelling van attributen.

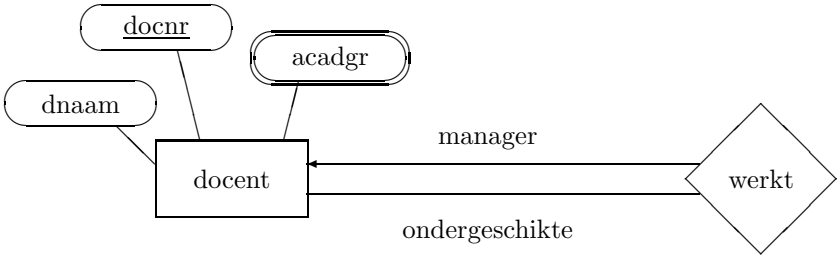
- **Ruiten**: voorstelling van relatie-verzamelingen.
- **Lijnen**: verbindingen van attributen aan entiteit-verzamelingen en entiteit-verzamelingen aan relatie-verzamelingen.

Bij elke component wordt ook de corresponderende naam weergegeven.
Een voorbeeld van een E-R diagram is weergegeven in figuur 14. Dit diagram bestaat uit twee entiteit-verzamelingen (*student* en *richting*) en één binaire relatie (*StRi*). In deze figuur wordt de relatie als een veel-op-één voorgesteld door middel van de pijl tussen de relatie en de entiteit *richting*.
Figuur 15 is een voorbeeld van een één-op-één relatie: elke student heeft slechts één biografie en een biografie hoort bij één welbepaalde student.



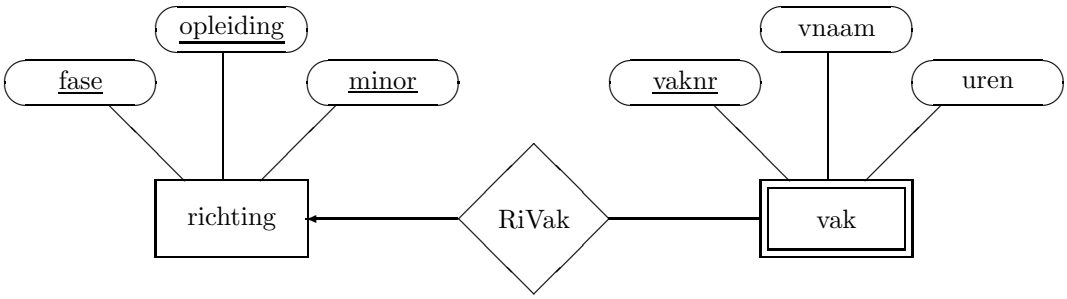
Figuur 15: Eén op één relatie

Rollen kunnen op een E-R diagram weergegeven worden door middel van een label bij de lijnen tussen de rechthoeken en de ruiten (figuur 16).

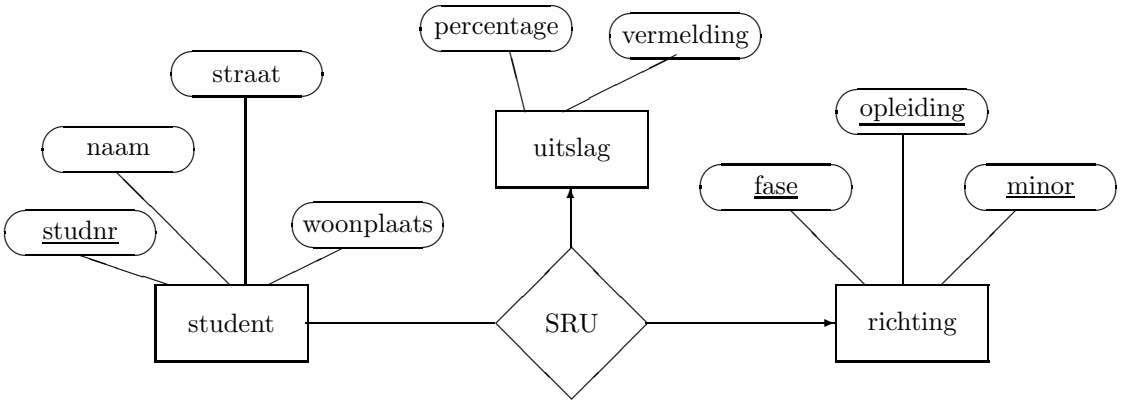


Figuur 16: E-R diagram met rol indicatoren

Een zwakke entiteit-verzameling wordt op een E-R diagram aangegeven met een dubbel omlijnde rechthoek. De relatie die de entiteit verbindt met de sterke entiteit waarop de primaire sleutel gevormd wordt, wordt aangegeven met een vettere lijn (figuur 17).



Figuur 17: E-R diagram met een zwakke entiteit verzameling



Figuur 18: E-R diagram met ternaire relatie

Ook niet-binaire relatieverzamelingen kunnen gemakkelijk weergegeven worden in een E-R diagram, bijvoorbeeld *SRU* in figuur 18.

2.7 Herleiden van ER-diagrammen naar tabellen

Een databank die overeenkomt met een E-R diagram kan voorgesteld worden door een verzameling van tabellen. Voor elke entiteit-verzameling en elke relatie-verzameling in de databank, bestaat er een unieke tabel. Elke tabel bestaat uit een aantal kolommen met unieke namen.

Sterke entiteit-verzamelingen.
E, een sterke entiteit-verzameling met attributen $a_1, a_2, \ldots, a_n$, kan voorgesteld worden met een tabel met naam E met n verschillende kolommen overeenkomend met de verschillende attributen van E. Elke rij in deze tabel correspondeert met een entiteit uit de entiteit-verzameling E.
Een nieuwe entiteit aan de databank toevoegen komt overeen met het toevoegen van een nieuwe rij in de tabel. Rijen kunnen ook verwijderd of aangepast worden.
In tabel 2 wordt de tabel *docent* voorgesteld. Deze tabel heeft twee kolommen overeenkomend met de twee attributen van de entiteit. Voor de eenvoud is het derde (multi-valued) attribuut, *acadgr*, weggelaten.

docnr	*dnaam*
71012	Nemegeer
72031	Appels
73025	Van Nuffelen
74016	Van Wunsel
75009	Peeters
76005	De Weerdt
77019	Pinxten

Tabel 2: De *docent* tabel

D_1 is de verzameling van alle docent-nummers en D_2 is de verzameling van alle docent-namen. Elke rij uit de *docent* tabel bestaat uit een 2-tuple (v_1, v_2) waarbij v_1 een docent-nummer is ($v_1 \in D_1$) en v_2 een docent-naam ($v_2 \in D_2$). Gewoonlijk zal de *docent* tabel slechts een deelverzameling bevatten van alle mogelijke rijen. De verzameling van alle mogelijke rijen van *docent* wordt het **cartesisch produkt** van D_1 en D_2 genoemd, met notatie

$$D_1 \, X \, D_2 \qquad \text{of} \qquad X_{i=1}^{2} \, D_i$$

Zwakke entiteit-verzamelingen.
A is een zwakke entiteit-verzameling met beschrijvende attributen $a_1, a_2, \ldots, a_r$. B is de sterke entiteit-verzameling waarvan A afhankelijk is. De primaire sleutel van B bestaat uit $b_1, b_2, \ldots, b_s$. A kan voorgesteld worden met een tabel met naam A met kolommen overeenkomend met de verschillende attributen uit de verzameling $\{a_1, a_2, \ldots, a_r\} \cup \{b_1, b_2, \ldots, b_s\}$

fase	*opleiding*	*minor*	*vaknr*	*vnaam*	*uren*
3	c	c	1	analytische chemie	4
3	c	c	2	organische chemie	3
3	e	i	1	analoge transmissie	3
3	e	i	2	databanken	2
3	e	i	3	web-technologie	4
3	em	em	1	toeg. elektronica	4
3	em	em	2	materiaalkeuze	3

Tabel 3: De *vak* tabel

In tabel 3 wordt de tabel *vak* voorgesteld. Omdat *vak* ondergeschikt is aan *richting* moet de primaire sleutel van deze sterke entiteit mee opgenomen worden in de tabel.

Relatie-verzamelingen.
R is een relatie-verzameling tussen entiteit-verzamelingen $E_1, E_2, \ldots, E_n$. Stel, primaire-sleutel(E_i) is de naam van de verzameling attributen die de primaire sleutel vormen van de entiteit-verzameling E_i. Indien R geen beschrijvende attributen heeft, dan wordt de tabel overeenkomend met R gevormd uit de kolommen:

$$\bigcup_{i=1}^{n} \text{primaire-sleutel}(E_i)$$

Indien R wel beschrijvende attributen $(a_1, a_2, \ldots, a_m)$ heeft, dan wordt de tabel overeenkomend met R gevormd uit de kolommen:

$$\bigcup_{i=1}^{n} \text{primaire-sleutel}(E_i) \cup \{a_1, a_2, \ldots, a_m\}$$

studnr	*fase*	*opleiding*	*minor*	bis
88163	3	e	e	
88234	3	e	e	
88356	3	e	i	1
90002	3	em	ae	1
92421	3	em	ae	
96375	3	c	b	

Tabel 4: De *StRi* tabel

In tabel 4 wordt de tabel *StRi* voorgesteld.

2.8 Generalisatie en specialisatie

De entiteit-verzameling *docent* kan uitgebreid worden door elke docent te klasseren als één van de volgende:

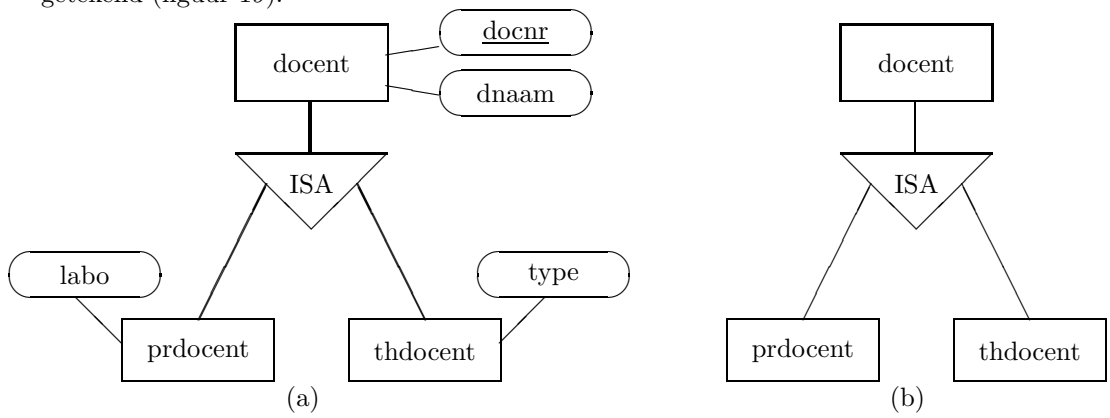
- *theorie-docent* (thdocent)

- *praktijk-docent* (prdocent)

Elk van deze entiteit-verzamelingen worden beschreven door een verzameling attributen die alle attributen van de entiteit-verzameling *docent* omvat plus een aantal additionele attributen. Bijvoorbeeld, de *praktijk-docent* entiteiten kunnen bijkomend beschreven worden door het *labo* attribuut. Voor *theorie-docent* kan het *type* attribuut bij opgenomen worden, om aan te geven of het over algemene, wetenschappelijke of technische vakken gaat.
De gemeenschappelijkheid van een aantal attributen kan uitgedrukt worden als een **generalisatie** of een **specialisatie**. Dit zijn omsluitende relaties die kunnen bestaan tussen een *hoger-niveau* entiteit-verzameling en één of meer *lager-niveau* entiteit-verzamelingen.

- **Generalisatie** is het resultaat van het nemen van de unie van twee of meer (lager-niveau) entiteit-verzamelingen om een hoger-niveau entiteit-verzameling te produceren.
- **Specialisatie** is het resultaat van het nemen van een deelverzameling van een hoger-niveau entiteit-verzameling om een lager-niveau entiteit-verzameling te vormen.

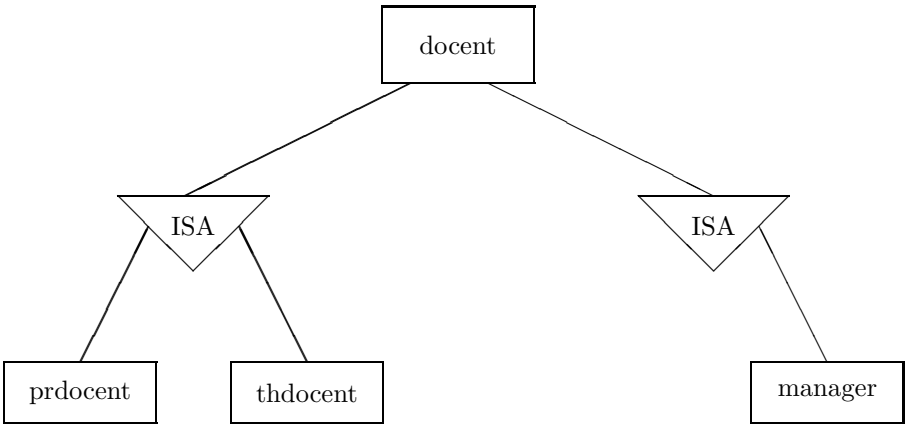
In het E-R diagram worden generalisatie en specialisatie voorgesteld door middel van een *driehoek* met label **ISA** ("is a"). Bij generalisatie worden de verbindingslijnen met de entiteiten vetter getekend (figuur 19).



Figuur 19: Generalisatie (a) en specialisatie (b)

Een klein verschil tussen generalisatie en specialisatie is de beperking bij generalisatie dat elke hoger-niveau entiteit ook een lager-niveau entiteit moet zijn. Bij generalisatie moet elke *docent* entiteit ofwel een *theorie-docent* ofwel een *praktijk-docent* zijn. Specialisatie laat de mogelijkheid toe dat een *docent* entiteit geen *theorie-docent* en ook geen *praktijk-docent* is, maar bijvoorbeeld een "adjunct-departementshoofd".
Generalisatie wordt gebruikt om de gelijkenissen tussen lager-niveau entiteiten te benadrukken en verschillen te verbergen. Specialisatie benadrukt het onderscheid tussen hoger-niveau en lager-niveau entiteit-verzamelingen. Een bijkomend aspekt is *attribuut-overerving*. De attributen van een hoger-niveau entiteit-verzameling worden geërfd door de lager-niveau entiteit-verzamelingen.
Het is mogelijk om tegelijk generalisatie en specialisatie in een E-R diagram te gebruiken. In figuur 20 wordt langs links aangegeven dat een docent ofwel praktijk- ofwel theoriedocent is; langs rechts wordt de mogelijkheid voorgesteld dat sommige docenten manager zijn.
Er zijn twee manieren om een E-R diagram met generalisatie of specialisatie om te vormen tot tabellen.

1. Creatie van een tabel voor elke hoger-niveau entiteit zoals hoger beschreven. Voor elke lager-niveau entiteit wordt een tabel gecreëerd met kolommen voor elke beschrijvende attribuut van de entiteit en kolommen voor elk attribuut van de primaire sleutel van de hoger-niveau entiteit. Uit het E-R diagram van figuur 19 worden 3 tabellen afgeleid:



Figuur 20: Combinatie van generalisatie en specialisatie

docent met attributen *docnr* en *dnaam*;

thdocent met attributen *docnr* en *type*;

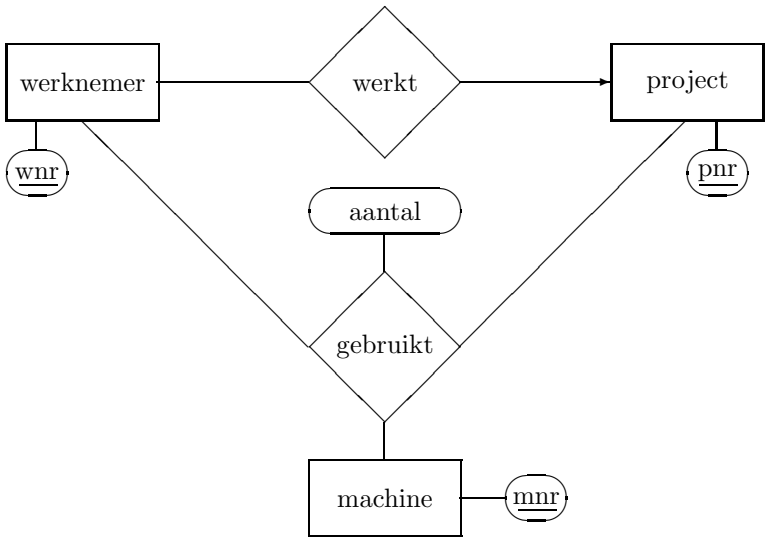
prdocent met attributen *docnr* en *labo*.

2. Er wordt geen tabel gecreëerd voor de hoger-niveau entiteit. Voor elke lager-niveau entiteit wordt een tabel gecreëerd met kolommen voor elke beschrijvende attribuut van de entiteit en kolommen voor *elk* attribuut van de hoger-niveau entiteit. Uit het E-R diagram van figuur 19 worden 2 tabellen afgeleid:

   thdocent met attributen *docnr*, *dnaam* en *type*;

   prdocent met attributen *docnr*, *dnaam* en *labo*.

2.9 Aggregatie

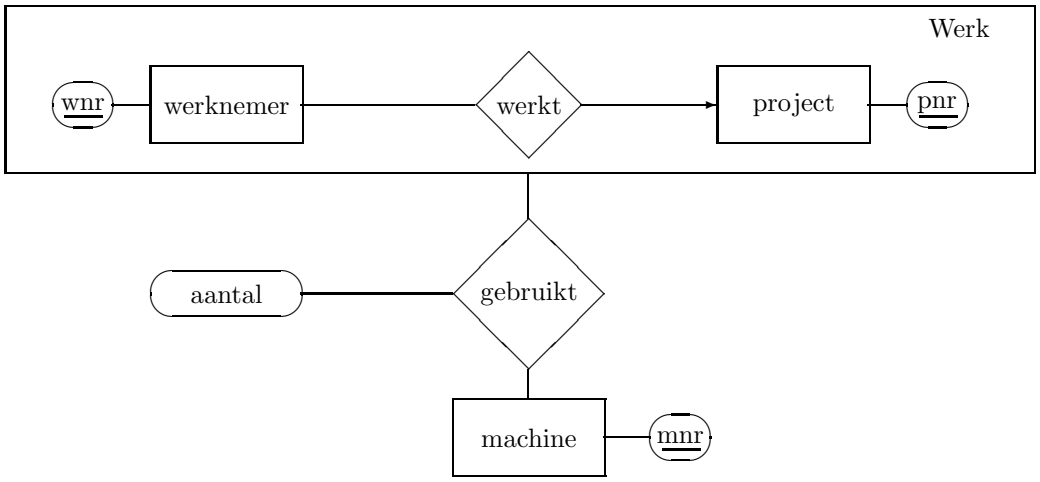


Figuur 21: E-R diagram met redundante relaties

Een beperking van een E-R model is dat het niet mogelijk is een relatie uit te drukken tussen relaties. Zo'n constructie is bijvoorbeeld nodig in een databank die informatie geeft omtrent werknemers die op een specifiek project werken en daarbij een aantal verschillende machines gebruiken. Met de basis E-R modelleringsconcepten wordt het E-R diagram van figuur 21 bekomen. De

relatie-verzamelingen *werkt* en *gebruikt* kunnen eventueel tot één relatie-verzameling gecombineerd worden. Maar daardoor wordt de logische structuur van het schema onduidelijk.

Aggregatie is een vorm van abstractie waardoor relatie-verzamelingen als hoger-niveau entiteiten beschouwd worden. In het voorbeeld kan de relatie-verzameling *werkt* en de entiteit-verzamelingen *werknemer* en *project* beschouwd worden als een hoger-niveau entiteit-verzameling *Werk*. Zo'n entiteit-verzameling wordt op dezelfde manier behandeld als een gewone entiteit-verzameling (bijvoorbeeld voorgesteld door een rechthoek zoals in figuur 22 te zien is).

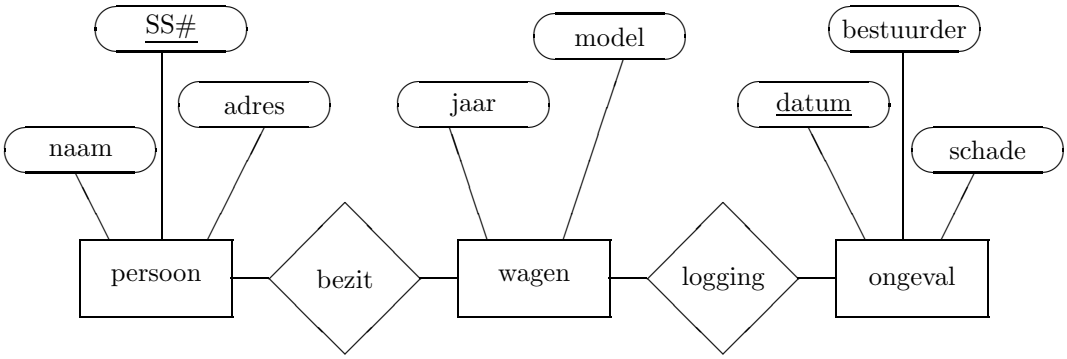


Figuur 22: E-R diagram met aggregatie

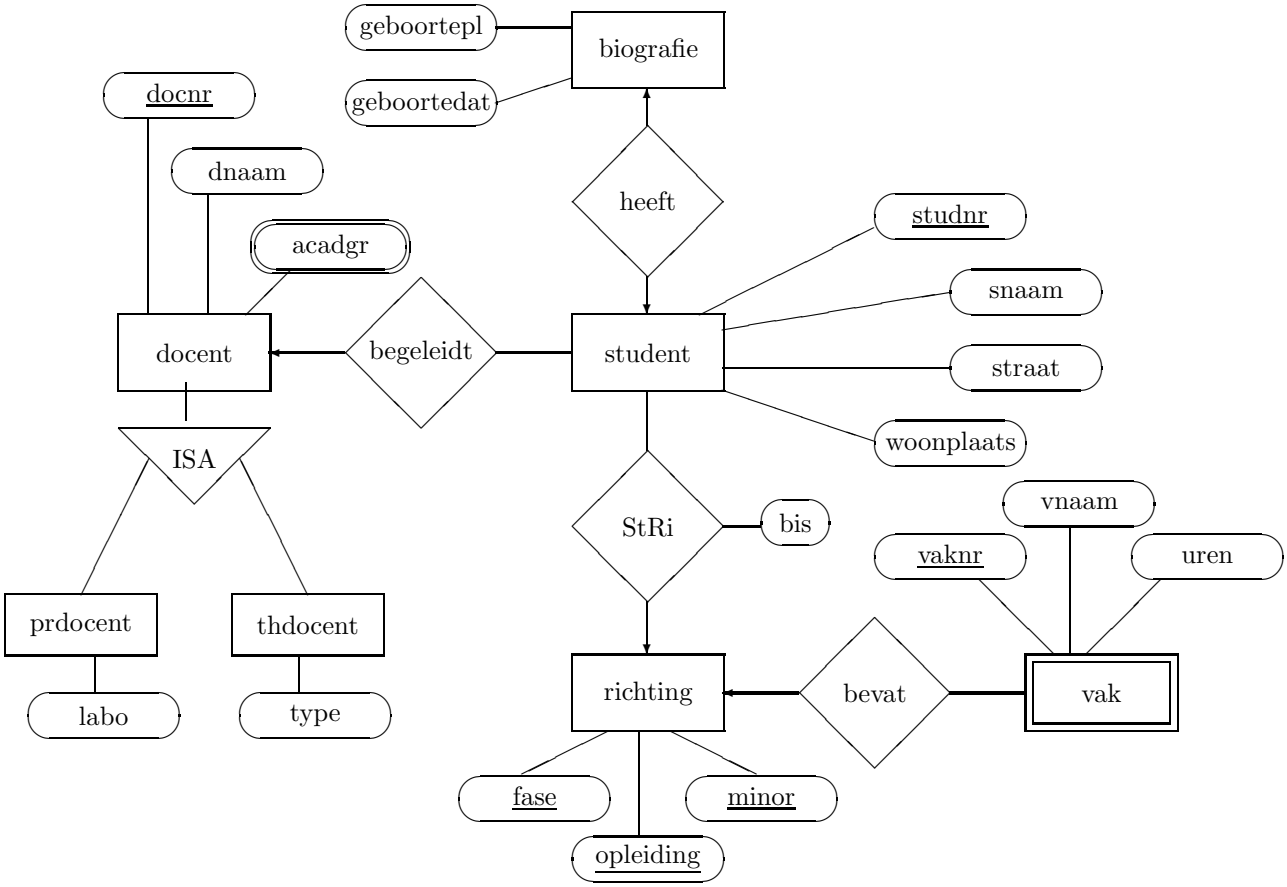
Een E-R diagram met aggregatie omvormen tot tabellen geeft geen problemen. Voor het voorbeeld moeten tabellen gecreëerd worden voor *werknemer*, *project*, *werkt*, *machine* en *gebruikt*. De tabel voor de relatie-verzameling *gebruikt* omvat een kolom voor elk attribuut uit de primaire sleutel van de entiteit-verzameling *machine* en van de entiteit *Werk*. Ook is een kolom opgenomen voor het beschrijvende attribuut (*aantal*) van de relatie *gebruikt*.

2.10 Voorbeelden

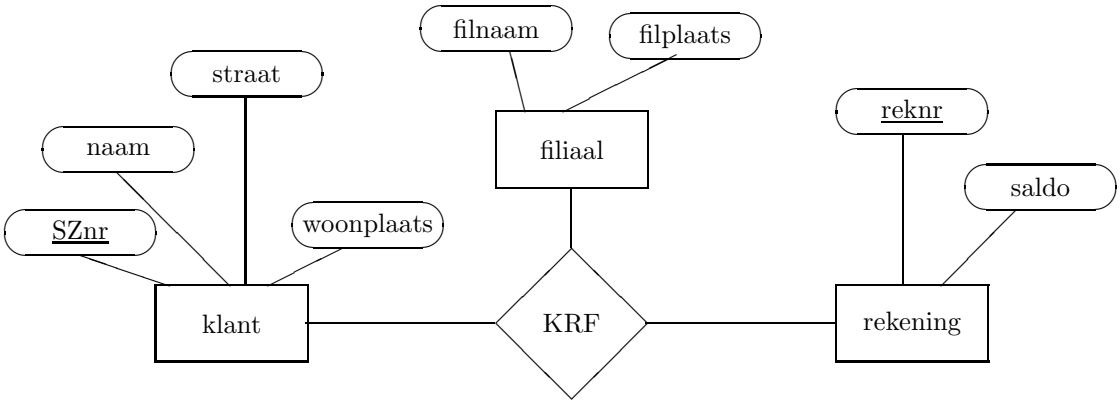
In figuur 23 is het E-R diagram getekend van een auto verzekeringsmaatschappij met een verzameling klanten die elk een aantal wagens bezitten. Voor welke wagen wordt een lijst van ongevallen bijgehouden.
In figuur 24 is het E-R diagram van `student` waarbij relationships aangegeven worden met een aantal andere entiteiten.
In figuren 25, 26 en 27 zijn E-R diagrams getekend van een klant met een rekening bij een bankfiliaal. Wat is het verschil tussen deze E-R diagrams?



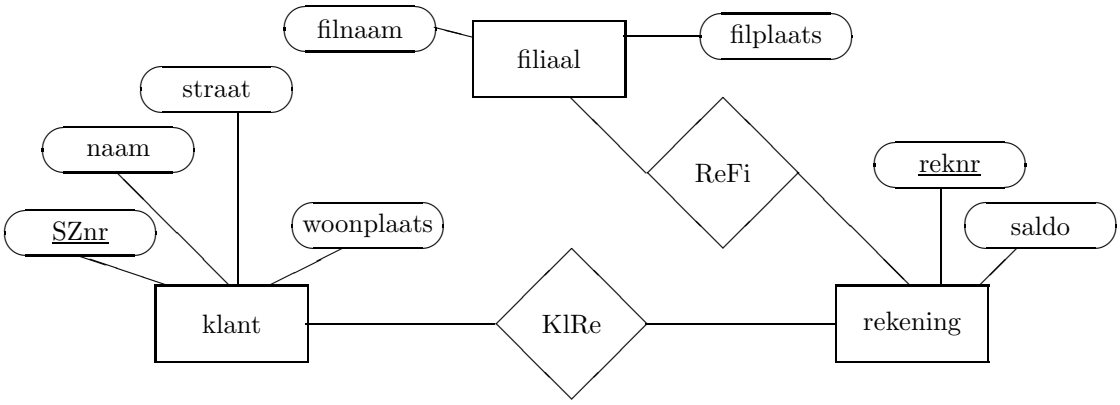
Figuur 23: E-R diagram: auto verzekeringsmaatschappij



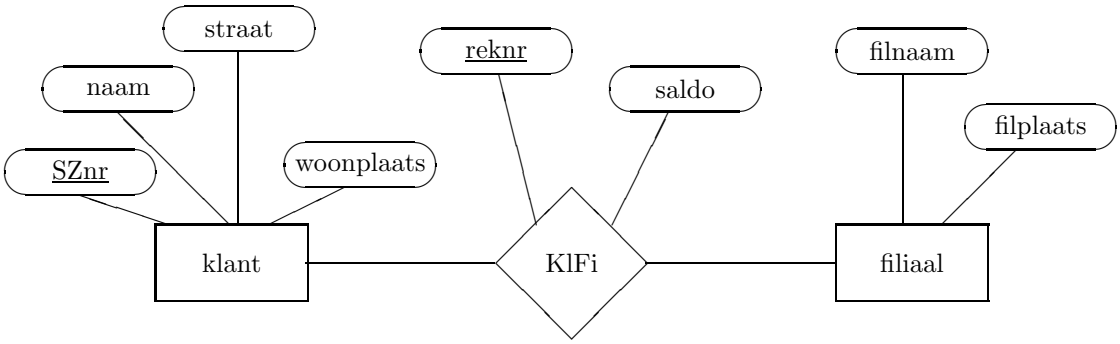
Figuur 24: E-R diagram: student met richting en vakken



Figuur 25: Klant met bankrekening in een bankfiliaal: versie 1



Figuur 26: Klant met bankrekening in een bankfiliaal: versie 2



Figuur 27: Klant met bankrekening in een bankfiliaal: versie 3

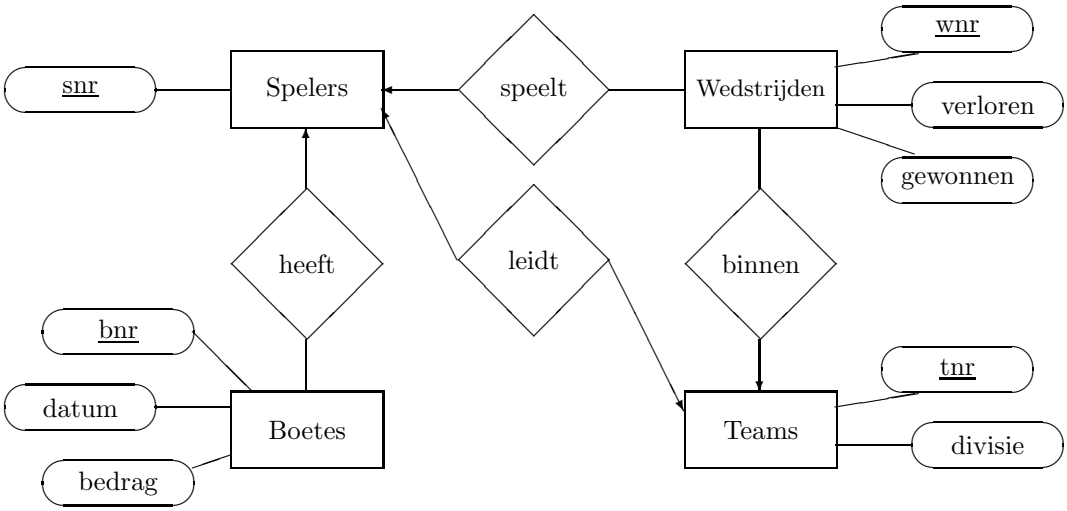
3 SQL: Data Definition/Manipulation Language

Data definition language wordt gebruikt voor het opzetten, aanpassen en verwijderen van tabellen. Data manipulation language wordt gebruikt voor het toevoegen, aanpassen en verwijderen van gegevens in deze tabellen.

3.1 Voorbeeld van een eenvoudige databank

In de voorbeelden van de SQL-statements in dit en volgend hoofdstuk wordt gebruik gemaakt van een database van een fiktieve tennisclub. Het ER diagram, weergegeven in figuur 28, bevat vier entiteitverzamelingen en relatieverzamelingen. Bij deze gegevens gelden volgende beperkingen.

- De aanvoerder van een team is zelf een speler. Een speler kan ten hoogste één team leiden.
- Elke speler in de wedstrijdentabel moet in de spelerstabel voorkomen en moet een bondsnummer hebben.
- Elk team in de wedstrijdentabel moet in de teamstabel voorkomen.
- Elke speler die boetes betaald heeft, moet in de spelerstabel voorkomen.
- Geboortejaar van een speler moet kleiner zijn dan het jaar van toetreding.
- Het jaar van toetreding moet groter zijn dan 1979. De datum van een boete moet ook in 1980 of daarna vallen.
- Elk boetebedrag moet groter dan nul zijn.



Figuur 28: ER diagram voor tennisclub

3.2 Maken van nieuwe tabellen

Met de instructie **create table** wordt een nieuwe tabel gecreëerd:

```
CREATE TABLE <tabelnaam>
             (  <kolomnaam> <datatype> [NOT NULL],
                ...
                PRIMARY KEY ( <kolommen-lijst> ),
                FOREIGN KEY ....                    )
```

Een kolomdefinitie bestaat uit een kolomnaam, een datatype en eventueel **not null**. Met behulp van een datatype geven we aan welk soort waarden een kolom kan bevatten. De verschillende datatypes zijn :

1. SMALLINT : Voor kleine, gehele getallen (tussen -32767 en 32767).
2. INTEGER : Voor grote, gehele getallen (tussen -2147483647 en 2147483647). Het woord integer mag afgekort worden tot INT.
3. DECIMAL : Hierbij kan men opgeven hoeveel cijfers voor en achter de komma geplaatst kunnen worden. Met bijvoorbeeld DECIMAL(12,4) wordt bedoeld 8 cijfers voor de komma en 4 achter de komma. De naam decimal mag afgekort worden tot DEC.
4. CHARACTER : Voor het opslaan van woorden, tekst en codes. Met het getal tussen de haakjes wordt de maximale lengte aangegeven (maximaal 255). Indien er niets wordt gespecificeerd is de lengte gelijk aan 1. De naam character mag afgekort worden tot CHAR.
5. DATE : Voor het definiëren van een datum.

Na het datatype kan de **not null**-optie gespecificeerd worden. Hierdoor mag een kolom geen null-waarden bevatten.

VB 1 : De spelerstabel is gemaakt met volgende instructies :

```
CREATE TABLE spelers ( snr     smallint not null,
                       naam    char(20) not null,
                       vl      char(5),
                       gbj     integer,
                       g       char check (g IN ( 'M', 'V') ),
                       ttj     integer check (ttj > 1979),
                       straat  char(20),
                       hnr     char(8),
                       pco     char(8),
                       plaats  char(20),
                       bdnr    smallint,
                       primary key (snr),
                       check ( ttj > gbj)
  );
```

Alternatief: `CREATE TABLE spelers ( snr   smallint PRIMARY KEY,`

3.3 Verwijderen van tabellen

We kunnen een tabel verwijderen met behulp van de instructie **drop table**.
VB 2 : De in VB 1 gemaakte tabel kan opnieuw verwijderd worden door :

```
DROP TABLE spelers PURGE
```

3.4 Beperkingen

3.4.1 Primaire sleutel

Waarschijnlijk is de belangrijkste type van beperking in een databank de declaratie dat een bepaald attribuut of een set van attributen een sleutel vormen voor een relatie.
Een primaire sleutel (= **primary key**) is een kolom of een verzameling kolommen van een tabel waarvan de waarden ten allen tijde uniek zijn. Null-waarden zijn niet toegestaan in de kolommen van een primaire-sleutel. Een primaire sleutel is noodzakelijk om geen twee identieke rijen op te slaan in een tabel. Hierdoor kan immers het probleem ontstaan dat de twee rijen niet meer van elkaar te onderscheiden zijn. Indien de primaire-sleutel uit meerdere kolommen bestaat, spreken we van een "samengestelde" primaire-sleutel. Bij het definiëren dienen deze gescheiden te worden door een komma.

3.4.2 Referentiële integriteit

Een tweede belangrijk type beperking is dat de waarde van bepaalde attributen zinvol moeten zijn, bijvoorbeeld ook voorkomen in een andere tabel. In SQL kan een attribuut of een set van attributen van een relatie gedeclareerd worden als een *foreign key*, refererend naar een attribuut of meerdere attributen van een tweede relatie (eventueel dezelfde relatie). Hierbij moet het gerefereerde attribuut (of attributen) van de tweede relatie wel de primary key zijn van deze relatie.
Wanneer de *foreign key* uit één attribuut bestaat, kan dit door na de attribuutnaam en het type iets toe te voegen:

```
snr     smallint   REFERENCES spelers,
```

Maar men kan ook een `FOREIGN KEY` beperking toevoegen. Bijvoorbeeld, bij de **teams tabel** kan gespecificeerd worden dat het tweede attribuut een waarde moet hebben die verwijst naar een bestaande speler in de **spelers** tabel.

```
CREATE TABLE teams ( tnr    smallint,
                     snr    smallint,
                     divisie  char(20),
                     PRIMARY KEY (tnr),
                     FOREIGN KEY (snr) REFERENCES spelers
                                       ON UPDATE CASCADE  );
```

Deze referentiële integriteit moet gegarandeerd blijven bij aanpassingen aan de databank. Er moet dus worden aangegeven wat er moet gebeuren indien

1. een rij in de **teams** tabel wordt toegevoegd met een *snr* waarde die niet voorkomt in de **spelers** tabel;
2. het *snr* veld in een rij in de **teams** tabel wordt aangepast met een waarde die niet voorkomt in de **spelers** tabel;
3. de speler uit de **spelers** tabel verwijderd wordt;
4. de waarde in de **spelers** tabel aangepast wordt.

Default : bij de default politiek van SQL wordt de aanpassing die niet conform de referentiële integriteit is, verworpen door het systeem, d.i. bij de vier hierboven opgesomde gevallen wordt een run-time fout gegenereerd.

CASCADE : bij verwijdering van een speler uit de **spelers** tabel (3) moeten alle referenties in de **teams** tabel verwijderd worden en bij aanpassingen aan een *snr* veld in **spelers** tabel (4) moeten alle referenties in **teams** tabel aangepast worden.

SET NULL : bij verwijderen (3) en aanpassen (4) in de **spelers** tabel wordt de waarde in de **teams** tabel op NULL gezet.

3.4.3 Waarden van attributen

Een eenvoudige beperking van deze soort is de **NOT NULL** beperking: rijen waarbij het specifieke attribuut de waarde NULL heeft, zijn niet toegestaan.
Een complexere beperking kan aan een attribuut gekoppeld worden met behulp van het CHECK keyword.

```
ttj      integer  CHECK ( ttj > 1979 ),
divisie  char(20)  CHECK ( divisie IN ( 'eerste', 'tweede', 'derde' ) ),
```

In praktijk gaat het meestal op een eenvoudige limiet op de waarde door middel van een opsomming of een rekenkundige beperking. In principe kan het wel elke mogelijke conditie zijn die in een WHERE kan gebruikt worden. In deze conditie kan gerefereerd worden naar het attribuut zelf of naar andere relaties of attributen van relaties. In dat geval moet de relatie wel in een FROM van de subquery vermeld worden. Bijvoorbeeld in de **teams** tabel:

```
snr   char(5)  CHECK ( snr IN ( SELECT snr FROM spelers ) ),
```

Dit lijkt op een referentiële integriteitsbeperking, maar is het niet:

- het toevoegen van een rij in de **teams** tabel met een *snr* waarde die niet voorkomt in de **spelers** tabel, mislukt (zoals eerder bij *default* uitgelegd);
- het aanpassen van een rij in de **teams** tabel met een nieuwe *snr* waarde die niet voorkomt in de **spelers** tabel, mislukt;
- maar, het verwijderen van een rij in de **spelers** tabel, zodat een bepaalde *snr* waarde verdwijnt, heeft geen effect op rijen in de **teams** tabel. Nogthans wordt na de verwijdering de CHECK beperking geschonden.

3.4.4 Globale beperkingen

Bij dit soort beperkingen worden verschillende attributen of zelfs verschillende tabellen met elkaar gekoppeld. De *tuple-based CHECK* beperkingen beperken een aspect van de rijen in één relatie. In de CREATE TABLE wordt na de lijst van attributen en de primary en foreign key specificaties een CHECK gevolgd door een voorwaarde tussen haakjes, toegevoegd. Deze voorwaarde kan weer alles zijn, wat na een WHERE kan geschreven worden, ook eventueel met inbegrip van subqueries. De voorwaarde van een tuple-based CHECK beperking wordt telkens wanneer een rij toegevoegd of aangepast wordt, gecontroleerd, en dit voor de nieuwe of aangepaste rij. Indien de voorwaarde niet geldt, is de beperking geschonden en de toevoeging of aanpassing wordt verworpen. Wanneer de conditie echter naar een andere relatie refereert en door een verandering in die andere relatie wordt de beperking geschonden, dan zal de CHECK de verandering toch niet beletten. Net als een attribuut-based CHECK is een tuple-based CHECK onzichtbaar bij andere relaties. Daarom is het best om in een tuple-based CHECK enkel attributen van het te controleren tuple te gebruiken, en geen subqueries. Hiervoor kan toch geen garantie gegeven worden, dat ze steeds blijven gelden.

```
CREATE TABLE A ( anr     char(5),
                 anaam   char(20),
                 kleur   char(8)
                     CHECK ( kleur IN ('rood','geel','groen','blauw') ),
                 gewicht integer  CHECK ( gewicht < 100 ),
                 pla     char(20),
                 PRIMARY KEY (anr),
                 CHECK ( kleur = 'blauw' OR gewicht > 20 )   );
```

3.5 Invoeren, wijzigen en verwijderen van gegevens.

Invoeren van gegevens in een tabel.

Met behulp van de **insert**-instructie kunnen gegevens in een tabel worden ingebracht.
VB 3 : De gegevens van een record kunnen ingebracht worden op volgende manier :

```
INSERT INTO spelers VALUES (6, 'Peeters', 'R', 1964,
       'M', 1977, 'Hazenlaan', '80', '2570', 'Duffel', 8467)
```

Indien er na de tabelnaam geen kolommen worden gespecificeerd, dient men voor elke kolom een waarde te geven. Indien de bijhorende waarde niet gekend is of niet van belang is, kan de waarde `NULL` gebruikt worden. Omgekeerd, indien men de kolommen wel specificeert, kunnen slechts een aantal waardes worden ingegeven. Ter illustratie geven we alleen het spelersnummer, de naam en het toetredingsjaar in :

```
INSERT INTO spelers (snr,naam,ttj) VALUES (100,'Wouters',2000)
```

Verwijderen van gegevens uit een tabel.
Met de **delete**-instructie kunnen we gegevens verwijderen.
VB 4 : We verwijderen de speler met spelersnummer gelijk aan 6.

```
DELETE FROM spelers
WHERE snr=6
```

Wijzigen van gegevens.
Met de **update**-instructie kunnen we gegevens wijzigen.
VB 5 : We veranderen de plaats van de speler met nummer 100 in Mechelen.

```
UPDATE spelers
SET plaats = 'Mechelen'
WHERE snr = 100
```

3.6 Inhoud van de tabellen

Naast de *spelers* en de *teams* tabel moeten ook nog de *boetes* en *wedstrijden* tabel gecreëerd worden.

```
CREATE TABLE boetes
(
     bnr    smallint,
     snr    smallint,
     datum  date,
     bedrag float,
     PRIMARY KEY (bnr),
     FOREIGN KEY (snr)
        REFERENCES spelers
        ON UPDATE SET NULL
        ON DELETE SET NULL
);
```

```
CREATE TABLE wedstrijden
(
    wnr       smallint,
    snr       smallint,
    tnr       smallint,
    gewonnen  smallint,
    verloren  smallint,
    PRIMARY KEY (wnr),
    FOREIGN KEY (snr) REFERENCES spelers
                      ON UPDATE CASCADE
                      ON DELETE CASCADE,
    FOREIGN KEY (tnr) REFERENCES teams
                      ON UPDATE CASCADE
                      ON DELETE CASCADE
);
```

Wanneer een speler uit de **spelers** tabel verwijderd wordt, wordt in de **boetes** tabel het overeenkomstige veld op NULL gezet: de informatie omtrent het boetebedrag verdwijnt niet.
Wanneer een speler uit de **spelers** tabel verwijderd wordt, worden in de **wedstrijden** tabel de wedstrijden die deze speler gespeeld heeft, ook verwijderd.
Het ER-diagram kan herleid worden tot vier tabellen: **spelers**, **teams**, **wedstrijden** en **boetes**. Merk op dat bijvoorbeeld de tabel die overeenkomt met de relatieverzameling *leidt*, in de **teams** tabel kan geintegreerd worden. Dit komt door de één-op-één relatie. Ook veel-op-één relaties kunnnen in een tabel van een entiteitsverzameling geintegreerd worden.

De SPELERS-tabel:

	SNR	NAAM	VL	GBJ	G	TTJ	STRAAT	HNR	PCO	PLAATS	BDNR
1	6	Peeters	R	1964	M	1979	Hazenlaan	80	2570	Duffel	8467
2	44	Bakker	E	1963	M	1980	Busstraat	23	2520	Ranst	1124
3	83	Hofland	PK	1956	M	1982	Mariakade	16	2570	Duffel	1608
4	2	Emonds	R	1962	M	1985	Bremweg	43	2570	Duffel	2411
5	27	Cools	DD	1964	V	1983	Liespad	84	2980	Zoersel	2513
6	104	Moerman	D	1970	V	1984	Stoutlaan	65	2980	Zoersel	7060
7	7	Wijers	GW	1963	M	1981	Musweg	39	2570	Duffel	?
8	57	Bohemen	M	1971	M	1985	Musweg	16	2570	Duffel	6409
9	39	Biskop	D	1956	M	1980	Wetsplein	78	2570	Duffel	?
10	112	Baalen	IP	1963	V	1984	Vosseweg	8	2470	Retie	1319
11	8	Nagels	B	1962	V	1980	Spoorlaan	4	2520	Ranst	2983
12	100	Peeters	P	1963	M	1979	Hazenlaan	80	2570	Duffel	6524
13	28	Cools	C	1963	V	1983	Oudebaan	10	2547	Lint	?
14	95	Meuleman	P	1961	M	1982	Hoofdweg	33	2531	Ranst	?

De TEAMS-tabel:

TNR	SNR	DIVISIE
1	6	eerste
2	27	tweede
3	39	derde

De WEDSTRIJDEN-tabel:

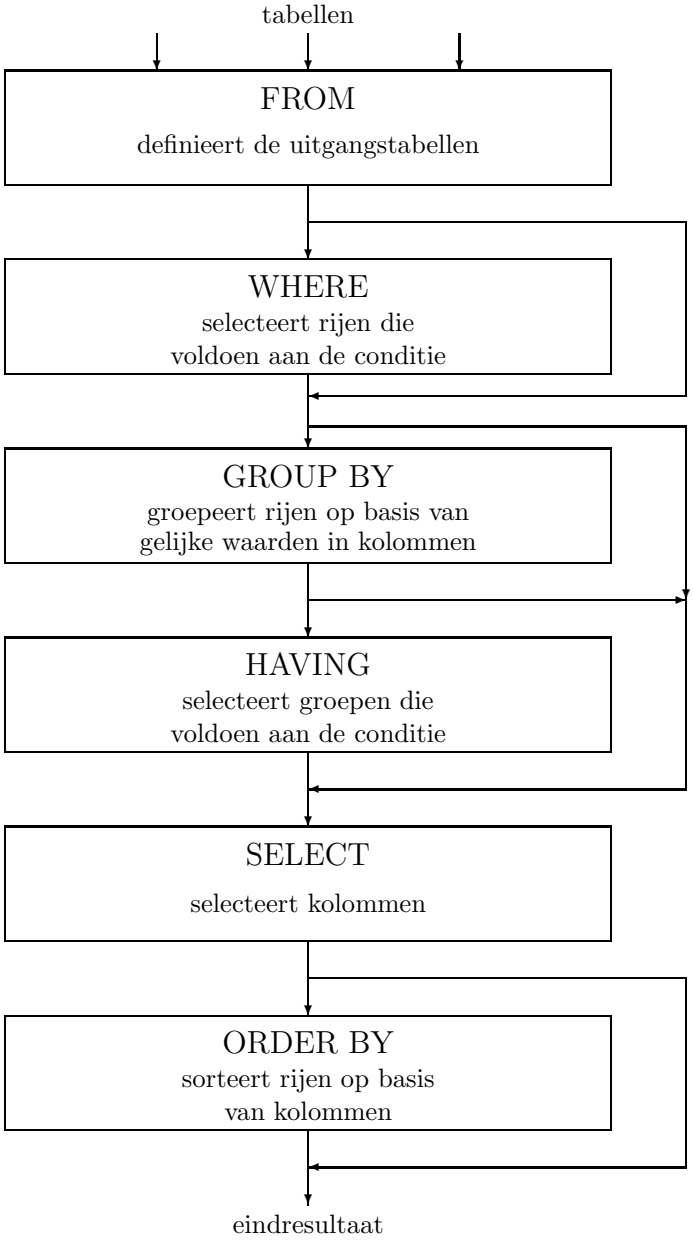
WNR	TNR	SNR	GEWONNEN	VERLOREN
1	1	6	3	1
2	1	6	2	3
3	1	6	3	0
4	1	44	3	2
5	1	83	0	3
6	1	2	1	3
7	1	57	3	0
8	2	8	0	3
9	2	27	3	2
10	2	104	3	2
11	2	112	2	3
12	2	112	1	3
13	2	8	0	3
14	3	44	3	0
15	3	39	1	3

De BOETES-tabel:

BNR	SNR	DATUM	BEDRAG
1	6	12/08/1980	100.00
2	44	05/05/1981	75.00
3	27	08/10/1983	100.00
4	104	12/08/1984	50.00
5	44	12/08/1980	25.00
6	8	12/08/1980	25.00
7	44	12/30/1982	30.00
8	27	11/12/1984	75.00

4 SQL: het vraagtaal gedeelte

4.1 Componenten van de SELECT-instructie



Figuur 29: Componenten van SELECT instructie

Bepaalde gegevens opvragen kan gebeuren d.m.v. SQL-statements. De structuur die hiervoor wordt gehanteerd ziet er als volgt uit :

(5) SELECT *attributen*
(1) FROM *tabellen*
(2) WHERE *conditie*
(3) GROUP BY *attributen*
(4) HAVING *conditie*
(6) ORDER BY *attributen*

De nummers tussen de haakjes duiden op de volgorde waarin het SQL-statement wordt uitgevoerd: figuur 29. Het gebruik van SQL zal aan de hand van voorbeelden verduidelijkt worden.

4.2 De FROM-component

VB 1 : GEEF DE NAMEN VAN DE SPELERS.

```
select naam, vl
from spelers
```

VB 2 : SELECTEER ALLE GEGEVENS UIT DE SPELERS-TABEL

```
select *
from spelers
```

VB 3 : HOEVEEL SPELERS ZIJN ER ?

```
select COUNT(*) as AANTAL
from spelers
```

VB 4 : WAT IS HET GEBOORTEJAAR VAN DE OUDSTE SPELER ?

```
select MIN(gbj)
from spelers
```

Naast **count** en **min** zijn er in SQL ook andere statistische functies mogelijk : **max**, **sum** en **avg**.

```
select MAX(bedrag), AVG(bedrag) from boetes
select SUM(gewonnen) from wedstrijden
```

VB 5 : ZOEK HET AANTAL DORPEN.

```
select count(plaats)
from spelers
```

Met **distinct** kan het aantal verschillende dorpen bepaald worden.

```
select count(distinct plaats)
from spelers
```

VB 6 : GEEF NAAM EN LEEFTIJD OP HET MOMENT VAN TOETREDEN.

```
select naam, ttj-gbj as leeftijd
from spelers
```

4.3 De WHERE-component

VB 7 : GEEF HET SPELERSNUMMER EN HET BONDSNUMMER VAN DIEGENEN DIE IN DUFFEL WONEN EN SORTEER VOLGENS BONDSNUMMER.

```
select snr, bdnr
from spelers
where plaats = 'Duffel'
order by bdnr
```

In hetgeen volgt zal stap voor stap besproken worden wat er gebeurt om tot het vereiste resultaat te komen.

1. FROM SPELERS : De gehele spelerstabel wordt geselecteerd.
2. WHERE PLAATS= 'Duffel' : Alleen de rijen met spelers die in Duffel wonen blijven over.

SNR	NAAM	...	PLAATS	BDNR
6	Peeters	...	Duffel	8467
83	Hofland	...	Duffel	1608
2	Emonds	...	Duffel	2411
7	Wijers	...	Duffel	?
57	Bohemen	...	Duffel	6409
39	Biskop	...	Duffel	?
100	Peeters	...	Duffel	6524

3. SELECT SNR, BDNR : Alleen de gevraagde gegevens (spelersnummer en bondsnummer) worden getoond.

SNR	BDNR
6	8467
83	1608
2	2411
7	?
57	6409
39	?
100	6524

4. ORDER BY BDNR : De gegevens worden geordend volgens bondsnummer.

SNR	BDNR
7	?
39	?
83	1608
2	2411
57	6409
100	6524
6	8467

OPMERKING : Zoals te zien is in de oplossing, worden null-waarden (`?`) altijd eerst geplaatst als er gesorteerd wordt.

De where-component maakt zeer dikwijls gebruik van **vergelijkingsoperatoren** om condities te testen.

```
=          gelijk aan
<          kleiner dan
>          groter dan
<=         kleiner dan of gelijk aan
>=         groter dan of gelijk aan
<>         verschillend van

      Voorbeelden :     PLAATS = 'Duffel'
                        GBJ < 1964
                        GBJ + 17 > TTJ
```

VB 8 : SELECTEER DE SPELERS MET EEN BONDSNUMMER

```
select snr, bdnr                    select snr, bdnr
from spelers                  of    from spelers
where bdnr = bdnr                   where bdnr IS NOT null
```

Meerdere condities kunnen gekoppeld worden met **and**, **or** en **not**.

VB 9 : ZOEK ALLE SPELERS DIE VROUW ZIJN OF DIE NA 1970 GEBOREN ZIJN.

```
select snr
from spelers
where g = 'V'  OR  gbj > 1970
```

VB 10 : ZOEK DE SPELERS DIE OF VAN DUFFEL OF IN 1963 GEBOREN ZIJN, MAAR NIET DE SPELERS DIE VAN DUFFEL EN IN 1963 GEBOREN ZIJN.

```
select snr
from spelers
where (plaats = 'Duffel' or gbj = 1963)
      and not  (plaats = 'Duffel' and gbj = 1963)
```

VB 11 : ZOEK DIEGENEN DIE TUSSEN 1962 EN 1964 GEBOREN ZIJN. (62 en 64 incl.)

```
select snr
from spelers
where gbj >= 1962 and gbj <= 1964
```

Opmerking. Het is efficiënter indien men deze where-instructie uitvoert met behulp van een **between** operator.

```
select snr
from spelers
where gbj BETWEEN 1962 and 1964
```

Met de **like**-operator kunnen we selecteren op alfanumerieke waarden met een bepaald patroon. Het **procentteken** en het **onderstrepingsteken** krijgen na 'like' een speciale betekenis. Het procentteken staat voor geen, één of meer tekens. Het **onderstrepingsteken** wordt gebruikt om juist één willekeurig teken voor te stellen.

VB 12 : ZOEK DE SPELERS WAARVAN DE NAAM BEGINT MET EEN 'B'.

```
select snr, naam
from spelers
where naam LIKE 'B%'
```

VB 13 : ZOEK DE SPELERS WAARVAN DE VOORLAATSTE LETTER EEN 'a' IS.

```
select snr, naam
from spelers
where rtrim(naam) like '%a_'
```

Om een conditie te testen waarbij een waarde moet worden vergeleken met een gegeven verzameling wordt gebruik gemaakt van de **in**-operator.

VB 14 : ZOEK ALLE PERSONEN DIE IN RANST, RETIE OF ZOERSEL WONEN.

```
select snr, naam, plaats
from spelers
where plaats IN ('Ranst','Retie','Zoersel')
```

4.4 Gegevens uit meerdere tabellen

VB 15 : GEEF HET TEAMNUMMER EN DE NAAM VAN DE AANVOERDER VAN ELK TEAM.

```
select tnr, naam
from teams, spelers
where teams.snr = spelers.snr
```

1. FROM TEAMS, SPELERS : De beide tabellen worden naast mekaar gezet.

TNR	SNR	DIVISIE	SNR	NAAM	...	PLAATS	BDNR
1	6	eerste	6	Peeters	...	Duffel	8467
1	6	eerste	44	Bakker	...	Ranst	1124
1	6	eerste	83	Hofland	...	Duffel	1608
1	6	eerste	2	Emonds	...	Duffel	2411
...	...	...	...	...	...	...	...
1	6	eerste	95	Meuleman	...	Ranst	?
2	27	tweede	6	Peeters	...	Duffel	8467
2	27	tweede	44	Bakker	...	Ranst	1124
2	27	tweede	83	Hofland	...	Duffel	1608
2	27	tweede	2	Emonds	...	Duffel	2411
2	27	tweede	27	Cools	...	Zoersel	2513
2	27	tweede	104	Moerman	...	Zoersel	7060
...	...	...	...	...	...	...	...
2	27	tweede	95	Meuleman	...	Ranst	?
3	39	derde	6	Peeters	...	Duffel	8467
...	...	...	...	...	...	...	...
3	39	derde	95	Meuleman	...	Ranst	?

   Opmerking. Elke rij van de spelerstabel wordt naast elke rij van de teamstabel gezet. We spreken ook van het *cartesisch product* van de betreffende tabellen.

2. WHERE TEAMS.SNR = SPELERS.SNR : De records met een zelfde spelersnummer in beide tabellen blijven over.

TNR	SNR	DIVISIE	SNR	NAAM	...	PLAATS	BDNR
1	6	eerste	6	Peeters	...	Duffel	8467
2	27	tweede	27	Cools	...	Zoersel	2513
3	39	derde	27	Biskop	...	Duffel	?

3. SELECT TNR, NAAM : Alleen de gevraagde gegevens blijven over.

TNR	NAAM
1	Peeters
2	Cools
3	Biskop

Om niet telkens de gehele tabelnaam te moeten gebruiken, kan men gebruik maken van (tijdelijke) afkortingen. De oplossing zou er dan als volgt uitzien :

```
select tnr, naam
from teams t, spelers s
where t.snr = s.snr
```

VB 16 : SELECTEER DE SPELERSNUMMERS VAN DE KAPITEINS DIE EEN BOETE HEBBEN BETAALD. ELKE NUMMER MAG SLECHTS 1 KEER VOORKOMEN.

```
select DISTINCT t.snr
from teams t, boetes bt
where t.snr = bt.snr
```

1. FROM TEAMS T, BOETES BT : selecteert de beide tabellen (cartesisch product).
2. WHERE T.SNR = BT.SNR : records met eenzelfde spelersnummer

TNR	SNR	DIVISIE	BNR	SNR	DATUM	BEDRAG
1	6	eerste	1	6	12/08/1980	100.00
2	27	tweede	3	27	08/10/1983	100.00
2	27	tweede	8	27	11/12/1984	75.00

3. SELECT DISTINCT T.SNR : Alleen de spelersnummers worden getoond. Het adjectief **distinct** zorgt ervoor dat identieke oplossingen slechts 1 maal worden getoond.

SNR
6
27

VB 17 : SELECTEER DE SPELERSNUMMERS VAN DE SPELERS DIE OUDER ZIJN DAN E. BAKKER.
Dit voorbeeld toon aan hoe men een vergelijking kan maken in eenzelfde tabel.

```
select s.snr
from spelers s, spelers p
where p.naam = 'Bakker' and p.vl = 'E'
      and s.gbj < p.gbj
```

De afkortingen `s` en `p` voor de spelerstabel zijn noodzakelijk.

4.5 GROUP BY en HAVING

VB 18 : GEEF HET SPELERSNUMMER VAN ELKE SPELER VOOR WIE MEER DAN 1 BOETE VAN MEER DAN 25.00 IS BETAALD EN SORTEER VOLGENS SPELERSNUMMER.

```
select snr
from boetes
where bedrag > 25.00
group by snr
having count(*) > 1
order by snr
```

In hetgeen volgt zal stap voor stap besproken worden wat er gebeurt om tot het vereiste resultaat te komen.

1. FROM BOETES : De gehele boetestabel wordt geselecteerd.

BNR	SNR	DATUM	BEDRAG
1	6	12/08/1980	100.00
2	44	05/05/1981	75.00
3	27	08/10/1983	100.00
4	104	12/08/1984	50.00
5	44	12/08/1980	25.00
6	8	12/08/1980	25.00
7	44	12/30/1982	30.00
8	27	11/12/1984	75.00

2. WHERE BEDRAG > 25.00 : Als resultaat blijven alleen die records over met een boete groter dan 25.00

BNR	SNR	DATUM	BEDRAG
1	6	12/08/1980	100.00
2	44	05/05/1981	75.00
3	27	08/10/1983	100.00
4	104	12/08/1984	50.00
7	44	12/30/1982	30.00
8	27	11/12/1984	75.00

3. GROUP BY SNR : De records worden gegroepeerd volgens spelersnummer.

BNR	SNR	DATUM	BEDRAG
1	6	12/08/1980	100.00
2	44	05/05/1981	75.00
7	44	12/30/1982	30.00
3	27	08/10/1983	100.00
8	27	11/12/1984	75.00
4	104	12/08/1984	50.00

4. HAVING COUNT(*) >1 : Alleen de spelers die meer dan één boete hebben blijven over.

BNR	SNR	DATUM	BEDRAG
2	44	05/05/1981	75.00
7	44	12/30/1982	30.00
3	27	08/10/1983	100.00
8	27	11/12/1984	75.00

5. SELECT SNR : Alleen het gevraagde attribuut per groep wordt getoond.

SNR
44
27

6. ORDER BY SNR : De spelersnummers worden gesorteerd in oplopende volgorde.

SNR
27
44

Met de group by-component kunnen we rijen groeperen op basis van overeenkomsten tussen die rijen. Vanaf dan kunnen alleen nog de groepen aangesproken worden, en niet meer de individuele rijen. De having-component heeft een vergelijkbare functie als de where-component. Met behulp van condities kunnen groepen geselecteerd worden. De having-component kan echter alleen in combinatie met de group by-component gebruikt worden.
Wanneer een groepering uitgevoerd wordt, kunnen bij de `select` alleen nog attributen vermeld worden die bij de `group by` staan of statistische functies op de groep. Ook een eventueel bijhorende `having` kan alleen attributen hebben die bij de `group by` staan of statistische functies op de groep.

VB 19 : GEEF PER WOONPLAATS HET AANTAL SPELERS.

```
select plaats, count(*)
from spelers
group by plaats
```

De functie *count(*)* wordt uitgevoerd op elke gegroepeerde rij.

PLAATS	COUNT(*)
Duffel	7
Lint	1
Ranst	3
Retie	1
Zoersel	2

VB 20 : GEEF VAN ELK TEAM HET TEAMNUMMER, HET AANTAL GESPEELDE WEDSTRIJDEN EN HET TOTAAL AANTAL GEWONNEN SETS.

```
select tnr, count(*), sum(gewonnen)
from wedstrijden
group by tnr
```

VB 21 : GEEF ALLE VERSCHILLENDE COMBINATIES VAN TEAMNUMMERS EN SPELERSNUMMERS UIT DE WEDSTRIJDEN TABEL.

```
select tnr, snr, count(*)
from wedstrijden
group by tnr, snr
```

Bij deze select-instructie wordt er gegroepeerd op twee kolommen. Alle rijen met hetzelfde teamnr en hetzelfde spelersnr vormen een groep. De volgorde van de kolomspecificaties in de group by-component heeft geen effect op het eindresultaat van een instructie.

TNR	SNR	count(*)
1	2	1
1	6	3
1	44	1
1	57	1
1	83	1
2	8	2
2	27	1
2	104	1
2	112	2
3	39	1
3	44	1

VB 22 : GEEF DE SPELERSNUMMER VAN ELKE SPELER DIE IN 1984 ZIJN LAATSTE BOETE HEEFT BETAALD.

```
select snr
from boetes
group by snr
having max(YEAR(datum)) = 1984
```

VB 23 : GEEF DE SPELERS DIE EEN TOTAAL BEDRAG AAN BOETES VAN 120 OF MEER HEBBEN BETAALD. GEEF TEVENS HET TOTAAL DAT REEDS BETAALD WERD.

```
select snr, sum(bedrag)
from boetes
group by snr
having sum(bedrag) > 120
```

BNR	SNR	DATUM	BEDRAG
1	6	12/08/1980	100.00
2	44	05/05/1981	75.00
5	44	12/08/1980	25.00
7	44	12/30/1982	30.00
3	27	08/10/1983	100.00
8	27	11/12/1984	75.00
4	104	12/08/1984	50.00
6	8	12/08/1980	25.00

2	44	75.00
5	44	25.00
7	44	30.00
3	27	100.00
8	27	75.00

44	130.00
27	175.00

4.6 De subquery

Een subquery is een select-instructie binnen een conditie van een select-instructie.
De subquery verschilt van een *gewone* select-instructie op twee vlakken :

- De select-component van een subquery mag nooit **distinct** bevatten.
- Een order by-component is niet toegestaan. De betekenis van een verzameling waarden verandert immers niet als dubbele waarden worden weggelaten of als de waarden anders worden gerangschikt.

Een subquery, gebruikt na < , > , = , ..., mag slechts 1 resultaat opleveren. Bij **exists**, IN, ALL en ANY varianten, mag de subquery resulteren in een verzameling waarden.
Een belangrijk aspect van de subquery is de reikwijdte van kolommen. Het is belangrijk te weten in welk gedeelte van de query een bepaalde tabel gekend is. Dit zal verduidelijkt worden met het volgende voorbeeld :

```
*****************************************************************Q1*********
*  select ...                                                              *
*  from  A                                                                 *
*                  ======================================Q2======         *
*  where ... in    =  (  select ...                              =         *
*                  =     from B                                  =         *
*                  =                  +++++++++++++++Q3++        =         *
*                  =     where ...    + (  select ...  +         =         *
*                  =                  +    from C  )+    )      =         *
*                  =                  ++++++++++++++++++        =         *
*                  ==============================================         *
*                                                                          *
*  and exists  (   ================Q4==                                     *
*                  =    select ...    =                                    *
*                  =    from D        =                                    *
*                  =    where ...     =   )                                *
*                  ====================                                    *
****************************************************************************
```

De tabel A is gekend in de blokken Q1, Q2, Q3 en Q4 en tabel B in de blokken Q2 en Q3.
De tabel C is alleen gekend in het blok Q3 en tabel D alleen in het blok Q4.

VB 24 : GEEF DE SPELERSNAAM VAN DE AANVOERDER VAN TEAM 1.

```
select naam, snr
from spelers
where snr = (  select snr
               from teams
               where tnr = 1  )
```

```
select naam, s.snr
from spelers s, teams
where s.snr = teams.snr
  and tnr = 1
```

De subquery mag slechts één resultaat geven (omwille van de test op gelijkheid '='). Indien de subquery meerdere resultaten geeft, kan er gebruik gemaakt te worden van de **in** operator.

VB 25 : ZOEK ALLE SPELERS DIE MINIMUM 1 WEDSTRIJD HEBBEN GESPEELD.

```
select snr, naam
from spelers
where snr in (  select snr
                from wedstrijden  )
```

```
select distinct s.snr, naam
from spelers s, wedstrijden w
where s.snr = w.snr
```

In deze where-instructie wordt gebruik gemaakt van een subquery, die in de wedstrijdentabel alle spelersnummers selecteert (6, 44, 83, 2, 57, 8, 27, 104, 112). Vermits deze selectie in een subquery gebeurt, worden alleen verschillende *snr*'s gegenereerd (DISTINCT is niet nodig).

OPMERKING. Een hoofdquery waarbij de = of **IN** operator gebruikt wordt op het resultaat van een subquery, kan steeds als een enkelvoudige query (mbv. join van tabellen) geschreven worden.

VB 26 : ZOEK DE SPELERS DIE EEN BOETE HEBBEN BETAALD.

```
select snr, naam
from spelers
where snr in (  select snr
                from boetes  )
```

```
select distinct spelers.snr, naam
from spelers, boetes
where spelers.snr = boetes.snr
```

VB 27 : HOEVEEL BOETES ZIJN GELIJK AAN DE LAAGSTE BOETE ?

```
select count(*)
from boetes
where bedrag = (  select min(bedrag)
                  from boetes  )
```

VB 28 : GEEF DE SOM VAN DE BOETES VAN DE SPELERS UIT RANST.

```
select SUM(bedrag)
from boetes
where snr in (  select snr
                from spelers
                where plaats = 'Ranst'  )
```

VB 29 : SELECTEER DE SPELERSNUMMERS VAN DE SPELERS DIE OUDER ZIJN DAN E. BAKKER, en geef ook het bijhorend geboortejaar.
Hiervoor is vroeger reeds een join query geschreven met behulp van de afkortingen s en p voor de spelerstabel. Men kan echter ook gebruik maken van een subquery :

```
select snr, gbj
from spelers
where gbj < (  select gbj
               from spelers
               where naam = 'Bakker' and vl = 'E'   )
```

Opmerking. Per rij uit de tabel van de hoofdquery wordt een subquery uitgevoerd. Deze subquery mag slechts één resultaat opleveren, omwille van de '<' test.
Beide queries resulteren in hetzelfde antwoord: `83, 2, 39, 8, 95`.

VB 30 : ZOEK DE OUDSTE SPELERS.

```
select *
from spelers
where gbj <= ALL (  select gbj
                    from spelers  )
```

Het tussenresultaat van de subquery bestaat uit de geboortejaren van alle spelers. In de hoofdquery wordt dan voor elke speler gekeken of zijn geboortejaar kleiner dan wel gelijk is aan elk geboortejaar in dit tussenresultaat.

VB 31 : ZOEK ALLE SPELERS DIE NIET TOT DE OUDSTE BEHOREN.

```
select *
from spelers
where gbj > ANY (  select gbj
                   from spelers  )
```

Het tussenresultaat bestaat wederom uit alle geboortejaren. Maar nu worden alle spelers gezocht met een geboortejaar dat groter is dan minstens één geboortejaar van één van de spelers. Als zo'n geboortejaar gevonden wordt, is de speler niet de oudste speler. Het eindresultaat van deze query bestaat uit alle spelers behalve de oudste (Hofman en Biskop).

VB 32 : GEEF HET NUMMER VAN ELKE SPELER MET MINSTENS 1 BOETE.

```
select snr, naam
from spelers
where exists (  select *
                from boetes
                where spelers.snr = boetes.snr  )
```

De gegevens van de spelerstabel zijn gekend in de gehele query. De gegevens van de boetestabel zijn alleen gekend in de subquery. Wanneer een voorwaarde in de WHERE van de subquery refereert maar een attribuut van een tabel gespecificeerd in de FROM van de hoofdquery, zijn deze twee queries *gecorreleerd*. Zo'n gecorreleerde query kan beter begrepen worden door te bedenken dat de geneste subquery geëvalueerd wordt voor elk tuple (of combinatie van tuples) van de hoofdquery.

1. FROM SPELERS : een kopie van de spelerstabel (eerste tussenresultaat)
2. WHERE EXISTS (...*subquery*...) : voor elke rij in dit eerste tussenresultaat wordt de subquery uitgevoerd. Het tussenresultaat van de subquery voor de eerste rij (snr = 6), geeft:

BNR	SNR	DATUM	BEDRAG
1	6	12/08/1980	100.00

   Er is één rij in de boetestabel waarin het spelersnummer gelijk is aan het spelersnummer van de rij in de spelerstabel. De conditie van de hoofdquery is waar, omdat het tussenresultaat van de subquery uit minstens één rij bestaat.

   Het tussenresultaat van de subquery voor de tweede rij uit de hoofdquery geeft drie rijen :

BNR	SNR	DATUM	BEDRAG
2	44	05/05/1981	75.00
5	44	12/08/1980	25.00
7	44	12/30/1982	30.00

   Het tussenresultaat van de subquery voor de derde rij uit de hoofdquery geeft geen enkele rij (speler 83 komt niet in de boetestabel voor). De `EXISTS` in de hoofdquery is dus niet waar; en er wordt verdergegaan met de vierde rij.
3. SELECT SNR, NAAM : Het uiteindelijke resultaat van de query is :

SNR	NAAM
6	Peeters
44	Bakker
27	Cools
104	Moerman
8	Nagels

VB 33 : GEEF HET TEAMNUMMER EN DE DIVISIE VAN ELK TEAM WAARIN SPELER 44 GESPEELD HEEFT.

```
select tnr, divisie
from teams
where exists (  select *
                from wedstrijden
                where snr = 44
                and tnr = teams.tnr  )
```

TNR	DIVISIE
1	eerste
3	derde

VB 34 : GEEF HET SPELERSNUMMER VAN ELKE SPELER MET MEER DAN 1 BOETE.

```
select distinct snr
from boetes bt
where snr in (  select snr
                from boetes
                where bnr <> bt.bnr  )
```

Voor elke rij in de boetestabel wordt gekeken of in deze tabel nog een rij aanwezig is met hetzelfde spelersnummer, maar met een ander boetenummer. Zo ja, dan heeft die speler minstens twee boetes.
Een andere mogelijke oplossing is :

```
select snr
from boetes
group by snr
having count(*) > 1
```

Beide mogelijkheden geven als resultaat :

SNR
44
27

VB 35 : GEEF DE SPELERS DIE GEEN WEDSTRIJD VOOR TEAM 1 HEBBEN GESPEELD.
Twee alternatieve oplossingen:

```
select snr, naam
from spelers
where snr not in ( select snr
                   from wedstrijden
                   where tnr = 1   )
```

```
select snr, naam
from spelers
where 1 <> ALL ( select tnr
                 from wedstrijden
                 where snr = spelers.snr )
```

VB 36 : GEEF VAN ELK TEAM WAARVAN DE AANVOERDER IN ZOERSEL WOONT, HET TEAMNUMMER EN HET AANTAL GESPEELDE WEDSTRIJDEN.

```
select tnr, count(*)
from wedstrijden
where tnr in ( select tnr
               from teams, spelers
               where teams.snr = spelers.snr
                     and plaats = 'Zoersel' )
group by  tnr
```

De subquery selecteert het nummer van het team waarvan de aanvoerder in Zoersel woont. De hoofdquery bepaalt daarna het aantal gespeelde wedstrijden. Dit geeft als oplossing :

TNR	COUNT(*)
2	6

VB 37 : GEEF DE DRIE HOOGSTE BONDSNUMMERS.

```
select bdnr
from spelers s1
where bdnr IS NOT NULL
      and 3 > (  select count(*)
                 from spelers s2
                 where s1.bdnr < s2.bdnr  )
order by bdnr DESC
```

BDNR
8467
7060
6524

De subquery telt voor elk bondsnummer het aantal bondsnummers dat groter is. Als dit aantal kleiner is dan drie, komt het betreffende bondsnummer in het resultaat. De functie **desc** zorgt ervoor dat de bondsnummers van groot naar klein worden geordend.

4.7 Combineren van select-instructies

Met behulp van de operators **union**, **intersect**, **except**, **union all**, **intersect all** en **except all** kunnen select-instructies met elkaar gecombineerd worden.

VB 38 : ZOEK DE SPELERS DIE EEN BOETE HEBBEN BETAALD OF DIEGENEN DIE AANVOERDER ZIJN.

```
select snr
from boetes
UNION
select snr
from teams
```

Het aantal attributen en de types van deze attributen van beide select instructies moeten gelijk zijn. Anders is er geen 'union' mogelijk. De select-componenten mogen geen 'distinct' bevatten. Door het gebruik van union verwijdert SQL automatisch alle dubbele rijen. Union is het equivalent van de operator *unie* of *vereniging* uit de verzamelingenleer.

VB 39 : ZOEK DE SPELERS DIE EEN BOETE HEBBEN BETAALD EN DIEGENEN DIE AANVOERDER ZIJN.

```
select snr
from boetes
INTERSECT
select snr
from teams
```

Intersect is het equivalent van de operator *intersectie* ofwel *doorsnede* uit de verzamelingenleer. Dubbele rijen worden ook hier uit het resultaat verwijderd.

VB 40 : ZOEK DE SPELERS DIE EEN BOETE HEBBEN BETAALD EN GEEN AANVOERDER ZIJN.

```
select snr
from boetes
EXCEPT
select snr
from teams
```

Except is het equivalent van de operator *verschil* uit de verzamelingenleer (in de Oracle databank wordt deze operator *MINUS* genoemd). Dubbele rijen worden ook hier verwijderd.

Opmerking. Indien het gewenst is dat dubbels niet verwijderd worden, gebruiken we **union all**, **intersect all** of **except all**.
De operaties INTERSECT en EXCEPT zijn **redundant** in SQL, ze kunnen ook op de volgende manier geschreven worden:

```
-- herschrijven intersect
       SELECT snr FROM boetes WHERE snr IN ( SELECT snr
                                             FROM teams  ) ;
-- herschrijven except
       SELECT snr FROM boetes WHERE snr NOT IN ( SELECT snr
                                                 FROM teams  ) ;
```

4.8 Select-instructie : de join

De join is een select-instructie waarbij ten eerste de from-component minstens twee tabellen bevat en ten tweede de where-component minstens één conditie bevat waarbij kolommen van verschillende tabellen met elkaar worden vergeleken.

VB 41 : SPELERSNUMMER VAN DE AANVOERDERS EN BIJHOREND TEAMNR:

```
select spelers.snr, tnr
from spelers, teams
where spelers.snr = teams.snr
```

De kolommen die in de WHERE-component van een SELECT-instructie voor de join zorgen, worden de *join-kolommen* genoemd. In het voorbeeld zijn dit *spelers.snr* (kolom 1 = K1) en *teams.snr* (kolom 2 = K2)
Tussen join-kolommen zijn vier soorten relaties mogelijk:

1. Populaties van K1 en K2 zijn *gelijk* : K1 = K2
2. De ene populatie is een *deelverzameling* van de andere: K1 $\subset$ K2 (of K2 $\subset$ K1)
3. De populaties zijn *conjunct* : K1 - K2 $\neq$ 0 and K2 - K1 $\neq$ 0
4. De populaties zijn *disjunct* : K1 - K2 = K1 and K2 - K1 = K2

Thetajoin. De algemene join of thetajoin heeft in SQL de volgende vorm :

```
select *
from spelers, teams
where spelers.snr THETA teams.snr
```

Indien de operator gelijk is aan '=', spreekt men van een **equijoin** :

VB 42 : GEEF VAN ELKE SPELER MET EEN BOETE DE PERSOONLIJKE GEGEVENS EN DE GEGEVENS I.V.M. DE BOETE.

```
select *
from spelers, boetes
where spelers.snr = boetes.snr
```

Dit geeft als resultaat :

SNR	NAAM	...	BDNR	BNR	SNR	DATUM	BEDRAG
6	Peeters	...	8467	1	6	12/08/1980	100.00
...	...	...	...	...	...	...	...
27	Cools	...	2513	8	27	11/12/1984	75.00

Indien gewenst is dat het spelersnummer slechts in 1 kolom wordt afgedrukt, maakt men gebruik van een **natuurlijke join** :

VB 43 : GEEF VAN ELKE SPELER MET EEN BOETE DE PERSOONLIJKE GEGEVENS EN DE GEGEVENS I.V.M. DE BOETE.

```
select spelers.*, boetes.bnr, boetes.datum, boetes.bedrag
from spelers, boetes
where spelers.snr = boetes.snr
```

Dit geeft als resultaat :

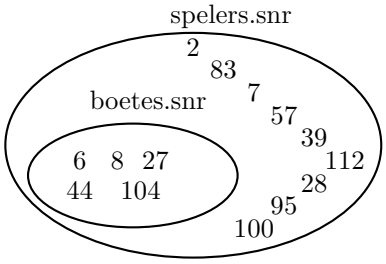
SNR	NAAM	...	BDNR	BNR	DATUM	BEDRAG
6	Peeters	...	8467	1	12/08/1980	100.00
...	...	...	...	...	...	...
27	Cools	...	2513	8	11/12/1984	75.00

VB 44 : GEEF VAN ELKE SPELER HET SPELERSNUMMER, DE NAAM EN HET BEDRAG VAN ZIJN BOETES.

```
select spelers.snr, naam, bedrag
from spelers, boetes
where spelers.snr = boetes.snr
```

Dit geeft als resultaat :

SNR	NAAM	BEDRAG
6	Peeters	100.00
44	Bakker	75.00
27	Cools	100.00
104	Moerman	50.00
44	Bakker	25.00
8	Nagels	25.00
44	Bakker	30.00
27	Cools	75.00



Voorgaande query wordt een **inner-equijoin** genoemd: alleen de gegevens over spelers die in de boetetabel voorkomen worden weergegeven.
Indien gewenst is dat gegevens over alle spelers worden gegeven, dient men gebruik te maken van een **outer-equijoin**. Dit kan gebeuren d.m.v. de "union"-instructie :

VB 45 : GEEF VAN ELKE SPELER HET NUMMER, DE NAAM EN HET BETAALDE BEDRAG.

```
select spelers.snr, naam, bedrag
from spelers, boetes
where spelers.snr = boetes.snr
union
select spelers.snr, naam, 0.00
from spelers
where snr not in (  select snr
                    from boetes  )
order by 3 desc
```

SNR	NAAM	BEDRAG
6	Peeters	100.00
27	Cools	100.00
27	Cools	75.00
44	Bakker	75.00
104	Moerman	50.00
44	Bakker	30.00
8	Nagels	25.00
44	Bakker	25.00
2	Emonds	0.00
7	Wijers	0.00
28	Cools	0.00
39	Biskop	0.00
57	Bohemen	0.00
83	Hofland	0.00
95	Meuleman	0.00
100	Peeters	0.00
112	Baalen	0.00

Join zonder join-conditie. Het begrip join werd gedefinieerd als een SELECT-instructie waar de FROM-component minimaal twee tabellen bevat en waarin de WHERE-component een voorwaarde bevat die kolommen van verschillende tabellen vergelijkt.
In het volgende voorbeeld is wel aan het eerste deel voldaan (FROM met twee tabellen) maar niet aan het tweede element (WHERE met een vergelijking van kolommen uit verschillende tabellen).

VB 46 : GEEF VOOR ELKE BOETE HET BOETENUMMER, HET BEDRAG PLUS DE SOM VAN BEDRAG EN DE BEDRAGEN VAN ALLE BOETES MET EEN LAGER BOETENUMMER (CUMULATIEVE WAARDE).

Ter vereenvoudiging gebruiken we volgende boetetabel :

BNR	SNR	DATUM	BEDRAG
1	6	12/08/80	100
2	44	05/05/81	75
3	27	09/10/82	100

```
select b1.bnr, b1.bedrag, sum(b2.bedrag)
from boetes b1, boetes b2
where b1.bnr >= b2.bnr
group by b1.bnr, b1.bedrag
order by b1.bnr
```

1. FROM boetes b1, boetes b2 : We laten alleen de bnr- en bedrag-kolom zien.

B1.BNR	B1.BEDRAG	B2.BNR	B2.BEDRAG
1	100	1	100
1	100	2	75
1	100	3	100
2	75	1	100
2	75	2	75
2	75	3	100
3	100	1	100
3	100	2	75
3	100	3	100

2. WHERE b1.bnr >= b2.bnr :

B1.BNR	B1.BEDRAG	B2.BNR	B2.BEDRAG
1	100	1	100
2	75	1	100
2	75	2	75
3	100	1	100
3	100	2	75
3	100	3	100

3. GROUP BY b1.bnr, b1.bedrag :

B1.BNR	B1.BEDRAG	B2.BNR	B2.BEDRAG
1	100	1	100
2	75	1	100
2	75	2	75
3	100	1	100
3	100	2	75
3	100	3	100

4. SELECT b1.bnr, b1.bedrag, SUM(b2.bedrag) :

B1.BNR	B1.BEDRAG	SUM(B2.BEDRAG)
1	100	100
2	75	175
3	100	275

5 Online analytical processing

Naast het gebruik van databanken in het nauwkeurig beheren van actuele gegevens in een operationele toepassing, kunnen database technieken ook gebruikt worden om strategische beslissingen te laten sturen op basis van informatie uit een databank.

5.1 Definities

OLTP : online transaction processing. De doelstelling is het onderhouden van een databank die een nauwkeurige weerspiegeling is van de actuele informatie-operaties van een onderneming. Het systeem moet in staat zijn voldoende aantal transacties per tijdseenheid te kunnen verwerken met een kleine responstijd om de belasting aan te kunnen zodat frustraties van gebruikers vermeden wordt. Karakteristieken:

- korte, eenvoudige transacties;
- frequente aanpassingen aan de gegevens;
- transacties die elk slechts een klein gedeelte van de databank aanspreken;
- hoge gebruiksgraad en dus hoge performatievereisten.

OLAP : online analytical processing. De doelstelling is het gebruik van informatie uit de databank om strategische beslissingen te ondersteunen (*decision support systems*): rapporteren van statistieken en trends. De gebruikte databanken zijn gewoonlijk zeer groot, maar hoeven niet altijd volledig nauwkeurig of up-to-date te zijn.

- complexe queries, korte responstijd minder belangrijk;
- bijna nooit aanpassingen aan de gegevens; de gegevens worden op geregelde tijdstippen ge-refreshed;
- transacties die elk een groot gedeelte van de databank aanspreken.

In de eerste OLAP toepassingen was de informatie in de OLAP databank afkomstig van de dagelijkse operaties die door de onderneming uitgevoerd worden, bijvoorbeeld uit de OLTP databank. In recentere toepassingen wordt actief op zoek gegaan naar bijkomende informatie die nodig is, bijvoorbeeld door deze aan te kopen. Zoals de *A* aangeeft, is het doel van een OLAP toepassing het *analyseren* van data. Hierin zitten twee gerelateerde elementen:

1. de analyse die moet uitgevoerd worden. Bijvoorbeeld, een bedrijf moet een beslissing nemen omtrent de mix van producten die in volgend boekhoudkundig kwartaal zullen geproduceerd worden. Hiervoor kan een analyse procedure ontworpen worden met als nodige input de verkoopcijfers van het vorige kwartaal en de historische verkoopcijfers uit equivalente periodes van de laatste vijf jaar.
2. de methodes om de grote hoeveelheden data die nodig is voor de analyse op een efficiënte manier te verwerven. Bijvoorbeeld, hoe kan het bedrijf de nodige verkoopcijfers extraheren uit de databanken van de verschillende departementen; in welke vorm moeten deze gegevens in de OLAP databank opgeslagen worden en hoe kunnen deze data efficient opgevraagd worden tijdens de analyse.

Het eerste element is geen databank probleem omdat hiervoor algoritmes nodig zijn die specifiek zijn voor het op te lossen probleem. In dit hoofdstuk wordt een inleiding gegeven op de mogelijke ondersteuning die een databank kan bieden bij zo'n analyse.
Nog enkele buzz-words:

Data warehouse : OLAP databanken worden gewoonlijk gestockeerd op speciale OLAP servers, die *data warehouses* genoemd worden. Deze hebben een speciale structuur om complexe OLAP queries te ondersteunen. Indien zo'n OLAP query in een OLTP omgeving zou uitgevoerd worden, zou deze de gewone OLTP handelingen gevoelig vertragen wat in een operationele omgeving ontoelaatbaar is. De term is in 1992 door Inmon geïntroduceerd:

"a subject-oriented, integrated, nonvolatile, time-variant collection of data in support of management's decisions".

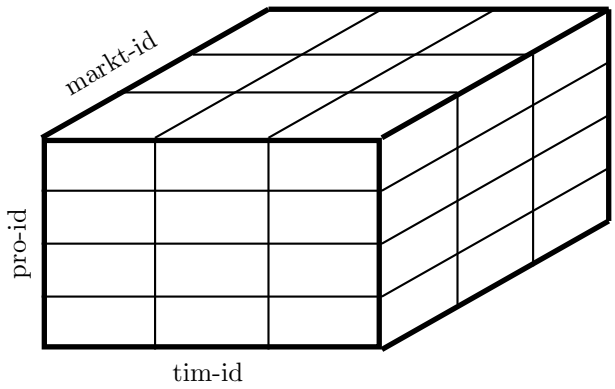
Data mart : een data warehouse met data die specifiek gericht is op een onderdeel van een organisatie (bijv. departement) of bedoeld is voor een specifiek aspect van de business analyse; data marts zijn dus stricter gefocusseerd.

Data mining : het doorzoeken van data met de intentie nieuwe *kennis* te ontdekken. Belangrijke doelstellingen zijn:

- *associatie*: het vinden van patronen in data waaruit regels kunnen afgeleid worden die de aanwezigheid van een verzameling items correleren met een bereik van waarden voor een andere verzameling items. Bijvoorbeeld: "wanneer een vrouw in een boetiek een handtas koopt, zal ze waarschijnlijk ook schoenen kopen".
- *classificatie*: het vinden van patronen in data om de data (en daarmee ook de items die door deze data beschreven wordt) te classificeren in een bepaald aantal interessante groepen. Bijvoorbeeld: een bedrijf zou zijn klanten willen kunnen classificeren als "groot-volume" kopers en "klein-volume" kopers. Toekomstige reklameinspanningen kunnen dan gerichter georganiseerd worden.
- *clustering*: bedoeling is ook te classificeren waarbij de categoriën door het clustering algoritme zelf ontdekt worden (bij classificatie worden de categoriën door de analyst zelf vooropgesteld).

5.2 Een multi-dimensioneel model

In een supermarktketen wil men een analyse maken van de hoeveelheid verkopen van verschillende producten in verschillende supermarkten over verschillende periodes. De verkoophoeveelheden (een gedeelte ervan) zijn weergegeven in tabel 5. `Markt-id`, `pro-id` en `tim-id` identificeren respectievelijk een specifieke supermarkt, een specifiek product en een specifieke periode. `Hoev` is de geldwaarde van de verkoop van dat product in die supermarkt over die periode. Zo'n tabel wordt een *fact* tabel genoemd omdat deze tabel alle feiten bevat omtrent de data die moet geanalyseerd worden.
Deze data kan als multi-dimensioneel gezien worden: de `markt-id`, `pro-id` en `tim-id` attributen zijn de dimensies en corresponderen met de argumenten van een functie. Het `hoev` attribuut correspondeert met de waarde van de functie. De data uit de fact tabel kan ook als multi-dimensionele *cube* voorgesteld worden. Voor het supermarktketen voorbeeld geeft dit een kubus (figuur 30) waarvan de dimensies gelijk zijn aan `markt-id`, `pro-id` en `tim-id` en de *cellen* van de kubus bevatten de corresponderende `hoev` waarden.



Figuur 30: Kubus: voorstelling met drie dimensies

De fact tabel:

Verkoop			
markt-id	pro-id	tim-id	hoev
M1	P1	T1	1000
M1	P2	T1	2000
M1	P3	T1	1500
M1	P4	T1	2500
M2	P1	T1	500
M2	P2	T1	800
M2	P3	T1	0
M2	P4	T1	3333
M3	P1	T1	5000
M3	P2	T1	8000
M3	P3	T1	10
M3	P4	T1	3300
M1	P1	T2	1001
M1	P2	T2	2001
M1	P3	T2	1501
M1	P4	T2	2501
M2	P1	T2	501
M2	P2	T2	801
M2	P3	T2	1
M2	P4	T2	3334
M3	P1	T2	5001
M3	P2	T2	8001
M3	P3	T2	11
M3	P4	T2	3301
M1	P1	T3	1002
M1	P2	T3	2002
M1	P3	T3	1502
M1	P4	T3	2502
M2	P1	T3	502
M2	P2	T3	802
M2	P3	T3	2
M2	P4	T3	333
M3	P1	T3	5002
M3	P2	T3	8002
M3	P3	T3	12
M3	P4	T3	3302

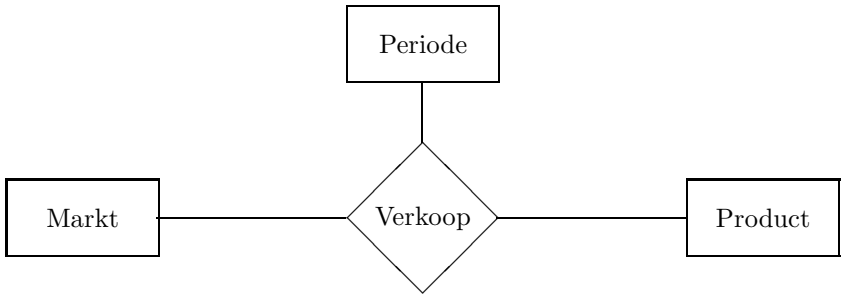
De dimensie tabellen:

Markt			
markt-id	stad	prov	gewest
M1	Lier	Antwerpen	Vlaanderen
M2	Tongeren	Limburg	Vlaanderen
M3	Spa	Luik	Wallonië

Product			
pro-id	naam	soort	prijs
P1	bier	drank	1.10
P2	zakdoekjes	zachtgoed	2.70
P3	hesp	vlees	3.90
P4	frisdrank	drank	1.05

Periode			
tim-id	week	maand	kwartaal
T1	1	januari	1
T2	23	juni	2
T3	51	december	4

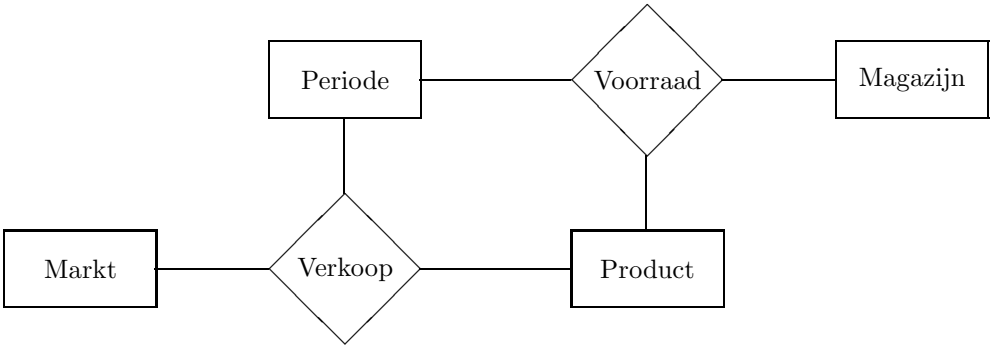
Tabel 5: Fact en dimensie tabellen



Figuur 31: Star schema

Bijkomende informatie omtrent de dimensies kan opgenomen worden in *dimensie*tabellen. Voor het supermarktketen voorbeeld zijn deze ook weergegeven in tabel 5. De `Markt` tabel beschrijft een markt, namelijk de stad, provincie en gewest. In een realistische toepassing zou de `Markt` tabel een rij bevatten voor elke supermarkt van de keten: dus verschillende markten in elke stad, verschillende steden in elke provincie, en verschillende provincies in elk gewest.
De relaties die in het supermarktketen voorkomen kunnen in een diagram voorgesteld worden, zoals weergegeven in figuur 31. Deze figuur heeft wat weg van een ster met de fact tabel in het centrum en de dimensietabellen in de stralen, vandaar de naam **star schema**. Een star schema correspondeert met een veel voorkomend fragment uit een ER-diagram, waarbij de fact tabel een relationship en de dimensietabellen entiteiten zijn.
Merk op dat de dimensietabellen NIET genormaliseerd zijn. Wanneer dit wel zou zijn, zouden er meer entiteiten getekend moeten worden en zou de figuur complexer worden (een *snowflake schema*). Hiervoor zijn twee redenen:

1. Dimensietabellen zijn klein ten opzichte van de fact tabel, zodat de gespaarde ruimte omwille van het elimineren van redundantie te verwaarlozen is.
2. Deze tabellen worden bijna nooit aangepast, zodat update anomalies geen issue zijn. Langs de andere kant zou de splitsing van de relaties tot heel wat overhead leiden bij queries.



Figuur 32: Constellation schema

Een OLAP toepassing kan uit verschillende fact tabellen bestaan die één of meer dimensie tabellen samen gebruiken. In ons supermarkt voorbeeld kan er een fact tabel `Voorraad` zijn met dimensie tabellen `Magazijn`, `Periode` en `Product`. De dimensie tabellen `Periode` en `Product` worden gedeeld met de `Verkoop` fact tabel. In figuur 32 wordt zo'n *constellation schema* voorgesteld.

5.3 Aggregatie

In heel veel OLAP queries is aggregatie een belangrijk onderdeel. Voorbeelden:

1. de totale verkoop van elk product in elke markt over alle periodes;
2. de totale verkoop van elk product over alle markten over alle periodes;
3. de totale verkoop in elke markt over alle producten over alle periodes;
4. de totale verkoop over alle producten over alle markten over alle periodes.

Deze verschillende totalen kunnen overzichtelijk in tabelvorm weergegeven worden. Omdat de aggregatie uitgevoerd is over alle periodes, wordt een gereduceerde view op de data geproduceerd: twee dimensies in plaats van drie.

totqty		markt-id			
		M1	M2	M3	totaal
pro-id	P1	3003	1502	15003	19509
	P2	6003	2403	24003	32409
	P3	4503	3	33	4539
	P4	7503	7000	9903	24406
	totaal	21012	10909	48942	80863

Deze resultaten kunnen met standaard SQL queries bekomen worden:

1. select pro-id, markt-id, sum(hoev) as totqty from Verkoop group by pro-id, markt-id;
2. select pro-id, sum(hoev) as totqty from Verkoop group by pro-id;
3. select markt-id, sum(hoev) as totqty from Verkoop group by markt-id;
4. select sum(hoev) as totqty from Verkoop;

Een nadeel is dat voor elke vorm van aggregatie een aparte query nodig is, die apart uitgevoerd wordt en een specifiek resultaat aflevert. Het zou interessant zijn

- om de verschillende niveaus van aggregatie in één enkele query op te vragen;
- een implementatie te hebben die alle gevraagde aggregaties tegelijk (en dus efficient) berekent.

In de SQL standaard van 1999 zijn een aantal opties toegevoegd aan de `GROUP BY`.
Met de `GROUPING SETS` optie kan de gebruiker precies aangeven welke groeperingen uitgevoerd moeten worden.

```
select markt-id, pro-id, sum(hoev) as totqty
from Verkoop
group by grouping sets ( (markt-id), (pro-id) );
```

markt-id	pro-id	totqty
M1	*null*	21012
M2	*null*	10909
M3	*null*	48942
null	P1	19509
null	P2	32409
null	P3	4539
null	P4	24406

Het systeem zal twee queries uitvoeren, eentje waarbij gegroepeerd wordt op `markt-id` en eentje waarbij gegroepeerd wordt op `pro-id`. Twee verschillende queries (in dit geval query 2 en 3) worden dus in één statement gebundeld, wat op zich niet erg is. Maar, SQL zal de *resultaten* van deze twee logisch verschillende queries ook bundelen in één tabel. En deze tabel is helemaal geen *relatie* (in de betekenis in een RDBMS omgeving). De `markt-id` rijen (met *null* in de `pro-id` kolom) hebben een totaal verschillende interpretatie van de `pro-id` rijen (met *null* in de `markt-id` kolom). De betekenis van `totqty` is afhankelijk van het voorkomen in een `markt-id` rij of in een `pro-id` rij. Ook de *null*s in de resultatentabel geven een andere soort van "ontbrekende informatie" aan. Zo'n *null* betekent hier duidelijk niet "waarde onbekend" of "waarde niet van toepassing" maar wat de betekenis dan wel juist is, is niet zo duidelijk. De gebruiker moet hiervoor een soort *rij-per-rij* denken aanwenden.

De `ROLLUP` optie is een verkorte vorm voor een `GROUPING SETS` combinatie

```
select markt-id, pro-id, sum(hoev) as totqty
from Verkoop
group by rollup ( markt-id, pro-id );
```

De `rollup` optie is logisch equivalent met

```
grouping sets ( (markt-id, pro-id),
                (markt-id), () );
```

De term `ROLLUP` komt van het feit dat de hoeveelheden "opgerold" worden langs de `markt-id` dimensie: eerst een groepering per markt en per product, dan een groepering per markt en tenslotte het totaal over alle markten samen.

markt-id	pro-id	totqty
M1	P1	3003
M1	P2	6003
M1	P3	4503
M1	P4	7503
M2	P1	1503
M2	P2	2403
M2	P3	3
M2	P4	7000
M3	P1	15003
M3	P2	24003
M3	P3	33
M3	P4	9903
M1	*null*	21012
M2	*null*	10909
M3	*null*	48942
null	*null*	80863

Met deze query worden query 1, 3 en 4 tegelijk geschreven. Bij het uitvoeren wordt eerst de aggregatie per markt en per product gedaan (query 1); deze sommen worden dan verder geaggregeerd om de totalen per markt te berekenen (query 3) en de som hiervan geeft dan de waarde die met query 4 zou berekend worden. Dit is heel wat efficiënter dan de drie queries onafhankelijk van elkaar uitvoeren.
De optie `ROLLUP` is niet symmetrisch: `group by rollup (markt-id, pro-id)` heeft een andere betekenis dan `group by rollup (pro-id, markt-id)`.

De `CUBE` optie is een verkorte vorm voor een andere `GROUPING SETS` combinatie. Met deze optie worden de vier queries in één command geschreven en bij de uitvoering wordt weer gebruik gemaakt van de resultaten van de meer specifieke aggregaties om de algemenere aggregaties uit te rekenen.

```
select markt-id, pro-id, sum(hoev) as totqty
from Verkoop
group by cube ( markt-id, pro-id );
```

De `cube` optie is logisch equivalent met

```
grouping sets ( (markt-id, pro-id),
                (markt-id), (pro-id), () );
```

Het resultaat geeft de verschilende groeperingen zowel langs de `markt-id` dimensie als langs de `pro-id` dimensie. Deze verschillende aggregatie worden met behulp van één query berekend. Een nadeel blijft de minder mooie rapportering in vergelijking met een tabelvorm (zie hoger).

markt-id	pro-id	totqty
M1	P1	3003
M1	P2	6003
M1	P3	4503
M1	P4	7503
M2	P1	1503
M2	P2	2403
M2	P3	3
M2	P4	7000
M3	P1	15003
M3	P2	24003
M3	P3	33
M3	P4	9903
M1	*null*	21012
M2	*null*	10909
M3	*null*	48942
null	P1	19509
null	P2	32409
null	P3	4539
null	P4	24406
null	*null*	80863

5.4 Hiërarchiën

In sommige dimensietabellen is een aggregatie hiërarchie aanwezig. Bijvoorbeeld in de markt tabel wordt aangegeven dat supermarkten zich in steden bevinden, dat steden in provincies liggen en dat provincies onderdeel zijn van gewesten.

markt-id → stad → prov → gewest

Queries kunnen op verschillende niveaus van deze hiërarchie uitgevoerd worden.

```
select pro_id, gewest, sum(hoev)
from Verkoop V, Markt M
where V.markt_id = M.markt_id
group by pro_id, gewest;
```

	Vlaanderen	Wallonië
P1	4506	15003
P2	8406	24003
P3	4506	33
P4	14503	9903

Wanneer een reeks van queries uitgevoerd wordt waarbij in de hiërarchie afgedaald wordt van het meer algemene naar het meer specifieke, spreekt men van **drilling down**. Om dit te kunnen doen is natuurlijk meer specifieke informatie nodig dan die vervat is in het resultaat van een meer algemene query. Dus om te kunnen aggregeren over provincies moet de fact tabel gebruikt worden of een eerder berekende tabel waarbij geaggregeerd is over steden.

```
select pro_id, prov, sum(hoev)
from Verkoop V, Markt M
where V.markt_id = M.markt_id
group by pro_id, prov;
```

	Antwerpen	Limburg	Luik
P1	3003	1503	15003
P2	6003	2403	24003
P3	4503	3	33
P4	7503	7000	9903

Het tegenovergestelde is **rolling up**: er wordt van het meer specifieke naar het meer algemene gegaan, dus opwaarts in de hiërarchie. Hier kan wel het resultaat van een meer specifieke query gebruikt worden om de meer algemene aggregatie uit te voeren. Bijvoorbeeld wanneer het resultaat van de vorige query zou bewaard worden in een tabel met naam `Prov-verkoop`, dan zou een roll up langs de markt hiërarchie kunnen gebeuren met de query;

```
select pro_id, gewest, sum(hoev) as totqty
from Prov-verkoop T, Markt M
where T.markt_id = M.markt_id
group by pro_id, gewest;
```

Bij *rolling up* kunnen dus eerder berekende resultaten gebruikt worden om op een efficiëntere manier resultaten te bekomen. En dit is een bijkomend reden voor het bestaan van de `ROLLUP` en `CUBE` opties bij de `GROUP BY`.

De data kan gezien worden als een multidimensionele cube; door een deelverzameling van de assen te selecteren wordt een **pivot** uitgevoerd (de multidimensionele cube wordt geheroriënteerd). De geselecteerde assen komen overeen met de lijst van attributen in de `group by`. Meestal wordt pivotering gecombineerd met een aggregatie over de overblijvende assen.
Bijvoorbeeld, een pivot op de multidimensionele cube om de data te zien vanuit de product en tijd dimensies. Het resultaat geeft de totale verkoop (over alle markten) voor elk product en voor elke maand.

```
select pro_id, maand, sum(hoev)
from Verkoop V, Periode P
where V.tim_id = P.tim_id
group by pro_id, maand;
```

	januari	juni	december
P1	6500	6503	6506
P2	10800	10803	10806
P3	1510	1513	1516
P4	9133	9136	6137

Er kan nu een *rollup* in de periode hiërarchie gebeuren door niet te groeperen op maanden maar op kwartalen.

```
select pro_id, kwartaal, sum(hoev)
from Verkoop V, Periode P
where V.tim_id = P.tim_id
group by pro_id, kwartaal;
```

	1	2	4
P1	6500	6503	6506
P2	10800	10803	10806
P3	1510	1513	1516
P4	9133	9136	6137

Men kan zich voorstellen dat de hiërarchie voor elke dimensie de multidimensionele cube onderverdeelt in subcubes. Bijvoorbeeld, het kwartaal niveau van de periode dimensie verdeelt de cube in subcubes, eentje voor elk kwartaal. Queries die informatie in verband met deze subcubes geven, worden **slicing and dicing** genoemd.

- Bij een pivot, d.i. gebruik van een `group by` om het niveau in de hiërarchie te specificeren, wordt de multidimensionele cube onderverdeeld in subcubes: alle elementen in het desbetreffende niveau worden samengenomen. Bijvoorbeeld, bij een `group by` op `pro-id` en `kwartaal` worden alle transacties voor hetzelfde product in hetzelfde kwartaal samen gegroepeerd. Dus, pivoteren creëert het effect van *dicing* de data cube in subcubes.
- Bij gebruik van een `WHERE` om een dimensie attribuut te vergelijken met een constante, wordt een specifieke waarde voor die dimensie gespecificeerd. Op die manier wordt een *slice* uitgevoerd.

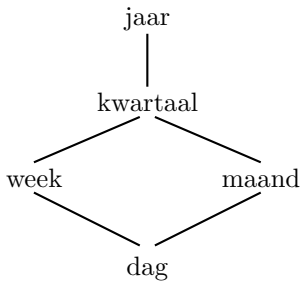
Meestal worden pivoting en slicing samen uitgevoerd en dit wordt dan "slicing and dicing" genoemd. Bijvoorbeeld, een query om de totale verkoop over alle markten in het eerste kwartaal te berekenen per product.

```
select pro_id, sum(hoev) as totqty
from Verkoop V, Periode P
where V.tim_id = P.tim_id and P.kwartaal = 1
group by pro_id;
```

pro-id	totqty
P1	6500
P2	10800
P3	1510
P4	9133

Alle bovenstaande queries gebruiken een groot deel van de data uit de fact tabel, wat typisch is voor een query in een data warehouse. Dit is in tegenstelling tot een OLTP query naar de databank van de lokale supermarkt, bijvoorbeeld *hoeveel dozen tomatensap zijn er in voorraad*, waarbij slechts één tuple aangesproken wordt.

Merk op dat een aggregatie hiërarchie niet lineair hoeft te zijn, zoals bij stad, provincie, gewest. De periode hiërarchie bijvoorbeeld is een *traliewerk*. Weken zitten niet volledig vervat in maanden: dezelfde week kan op het einde van een maand en in het begin van een volgende maand zitten. Dus kan er een roll up gebeuren van dagen in weken **of** in maanden, maar met weken kan alleen een roll up naar kwartalen gebeuren.



5.5 Implementatie issues

Een belangrijke technische karakteristiek, *OLAP toepassingen werken met zeer grote hoeveelheden data, die relatief statisch is en aanpasingen infrequent zijn*, is verantwoordelijk voor de gespecialiseerde implementatie technieken voor OLAP systemen. Bij veel van deze technieken worden partiële resultaten of indices op voorhand berekend, wat erg aangepast is wanneer de queries die de gebruikers zullen uitvoeren op voorhand gekend zijn, bijvoorbeeld wanneer ze ingebed zijn in een operationele OLAP toepassing.

1. Vooraf berekenen van dikwijls gebruikte aggregaties en deze bijkomend stockeren in de databank, bijvoorbeeld als materialized views. Omdat de data niet dikwijls gewijzigd wordt, is de overhead om deze aggregatie waarden te onderhouden klein.
2. Gebruik van indices die specifiek gericht zijn op de queries die zullen uitgevoerd worden. Omdat data updates infrequent zijn is de normale overhead bij het onderhoud van indices minimaal. Twee specifieke indices hiervoor zijn:
   - **join index**: een speciale index structuur voor het optimaliseren van een join van de relaties in een star schema;
   - **bitmap index**: voor het indiceren van een attribuut dat slechts een beperkt aantal waarden kan aannemen. Zo'n attributen komen frequent voor in OLAP toepassingen. Bijvoorbeeld, `gewest` in de `Markt` tabel heeft slechts drie mogelijke waarden, *Vlaanderen*, *Wallonië* en *Brussel*. Wanneer de `Markt` tabel 10000 rijen bevat, heeft een bitmap index op `gewest` met drie bit vectoren een stockageruimte behoefte gelijk aan 30000 bits, of ongeveer 4 Kbytes. Een index met deze grootte kan gemakkelijk in primair geheugen gestockeerd worden en kan op die manier voorzien in een snelle toegang tot de records met corresponderende waarden.

5.6 Laden van een data warehouse

Een data warehouse is een speciale database met daarin data voor OLAP en data mining. Zo'n data warehouse is gewoonlijk zeer groot, tot zelfs enkele terabytes. De data is afkomstig van eventueel

verschillende operationele databanken (voor OLTP toepassingen) met elk eigen schema's. Deze data moet geregeld vanuit deze operationele databanken geladen worden in de data warehouse (DW). Dit is geen triviale operatie waarbij twee belangrijke bewerkingen moeten uitgevoerd worden voordat de data in de DW kan geladen worden.

1. Transformatie: de data uit de verschillende bronnen moet naar een gemeenschappelijk formaat omgezet worden:
   - syntactisch: bijv. in het schema van het ene DBMS wordt de SIS nummer voorgesteld door het attribuut `SIS` (character string) terwijl in een ander DBMS `sisnr` (integer) gebruikt wordt
   - semantisch: bijv. in de ene DBMS zijn de verkopen per uur geaggregeerd terwijl in een andere DBMS er geen aggregatie gedaan is, maar de individule verkooptransacties gestockeerd zitten; en het kan dan zijn dat in de DW de verkooptotalen op dagbasis gevraagd zijn.
2. Cleaning: corrigeren van fouten, aanvullen van ontbrekende informatie; men zou er kunnen van uitgaan dat de data uit een OLTP databank correct is, maar de ervaring leert dat dit helemaal niet het geval is. Daarnaast kan er ook data van uit andere bronnen dan OLTP databanken gehaald worden, die bijvoorbeeld onderhevig zijn aan tikfouten.

Wanneer de bronnen relationele databanken zijn met schema's die voldoende gelijkaardig zijn aan die van de DW en wanneer er geen data cleaning nodig is, dan kan de extractie en de toevoeging met één enkel SQL statement gebeuren.
Voorbeeld: veronderstel dat de winkel "M1" van de grootwarenhuisketen een `SALES` tabel heeft met attributen `pro-id`, `tim-id` en `hoeveelheid`; elke record hierin geeft de verkoop van een bepaald product in een bepaalde periode. Nadat periode `T4` afgelopen is, kan dan de fact tabel uit de DW aangevuld worden met de verkoopsinformatie van markt `M1` in periode `T4`:

```
insert into Verkoop(markt-id, pro-id, tim-id, hoev)
         select 'M1', S.pro-id, S.tim-id, S.hoeveelheid
         from SALES S
         where S.tim-id = 'T4'
```

Wanneer data cleaning en transformatie nodig zijn, dan kan de data die moet geëxtraheerd worden uit de brondatabanken eerst voorgesteld worden als views. Een opkuis programma kan dan via deze views de data opvragen, zonder daarbij specifieke kennis te moeten hebben van elk individueel databank schema en daarop de nodige transformaties doen. Tot slot kan dan de aangepaste data toegevoegd worden aan de data warehouse.
Laden en updaten in een OLAP databank is een niet-triviale taak omwille van de grote datavolumes. Omwille van efficiëntie gebeurt het updaten gewoonlijk incrementeel. Verschillende delen van de databank worden op verschillende momenten aangepast. Dit kan wel als gevolg hebben dat de databank in een inconsistente toestand terecht komt: niet voldaan aan bepaalde integriteitsbeperkingen of geen exacte weerspiegeling van de huidige situatie in het bedrijf. Maar dit is meestal niet zo erg voor OLAP queries omdat deze meestal dienen voor overzichten (sommen, gemiddeldes, aantallen) te berekenen en zo'n overzicht wordt niet erg beïnvloed door een inconsistentie.

6 Information retrieval

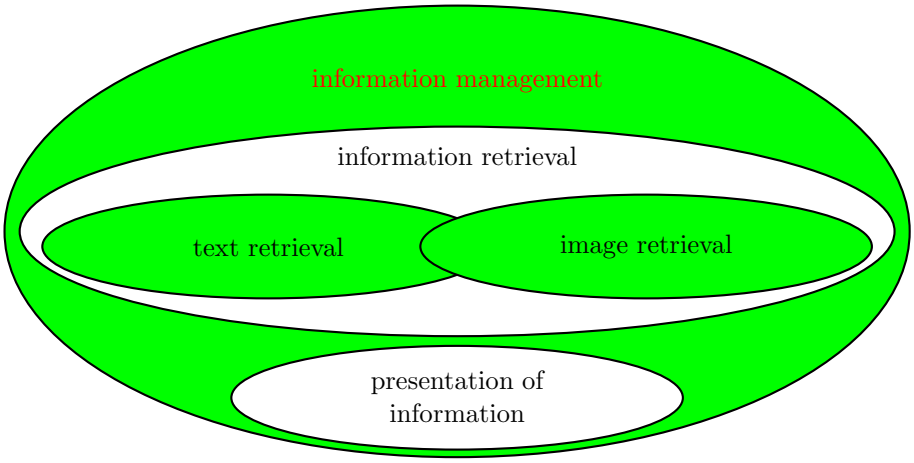
6.1 Situering

Definitie uit *Introduction to information retrieval*:

> *Information retrieval* (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

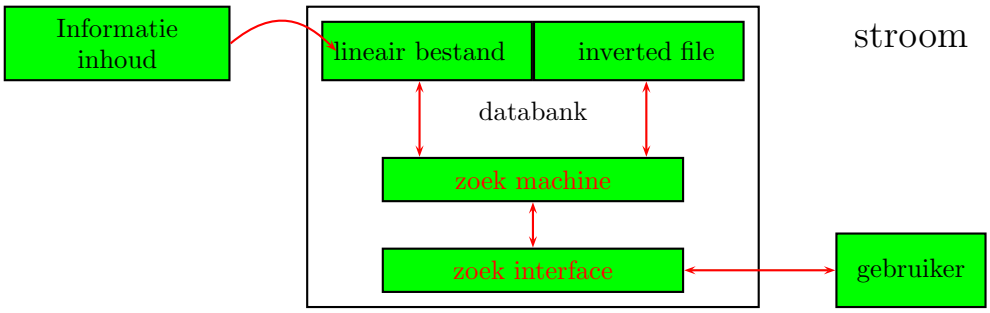
Een IR systeem geeft de gebruiker geen kennis op het vlak van zijn/haar onderzoeksdomein; het informeert alleen maar omtrent het bestaan (of niet-bestaan) van documenten gerelateerd aan het domein en waar deze te vinden zijn.
data retrieval:

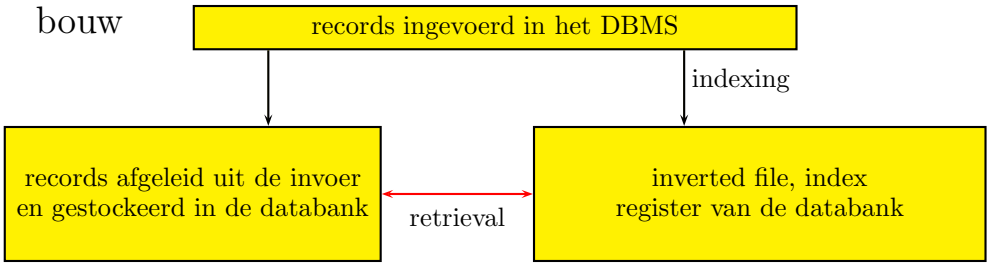
- zoeken naar een *exacte* overeenkomst: komt een item al of niet voor in een tabel (IR: documenten die *gedeeltelijk* overeenkomen met de zoekvraag, maar wel *relevant* zijn)
- *monothetische* classificatie: klassen bestaande uit objecten met attributen die nodig en voldoende zijn om tot die klasse te behoren (IR: *polythetisch*, geen attribuut is nodig of voldoende om tot een klasse te behoren)
- *artificiële* vraagtaal met een volledige specificatie van wat gevraagd is (IR: *natuurlijk* en als een gevolg geen exacte specificatie)



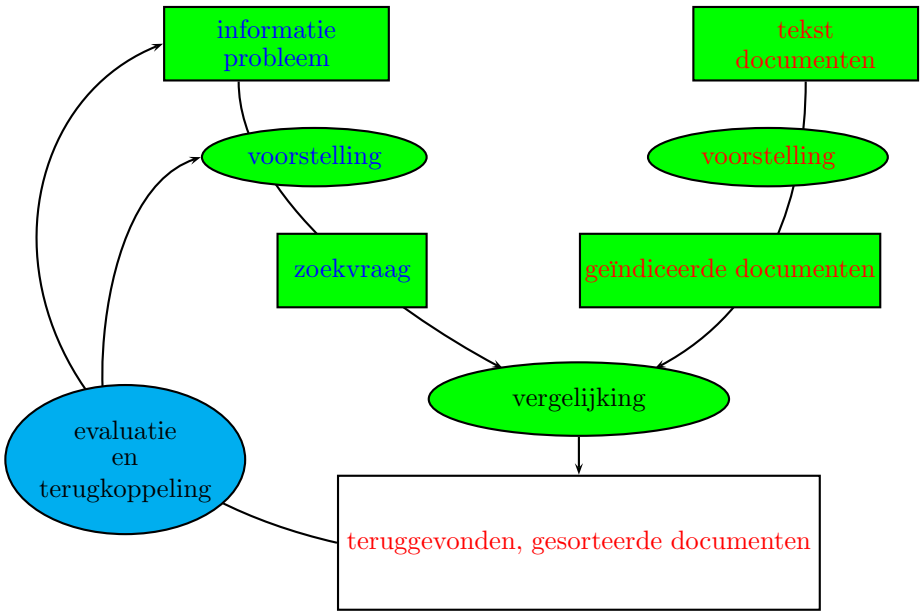
Overlap text/image retrieval: in zeer veel gevallen is het opzoeken van beelden niet gebaseerd op een gecomputeriseerde analyse van het beeld zelf, maar op het zoeken in de bijhorende tekst

6.1.1 Een tekst geöriënteerde databank





6.1.2 Basisprocessen in zoeksystemen



6.2 Indexing

Het voorbereiden van documenten om deze in een IR systeem te gebruiken: omvorming naar een *gemakkelijk toegankelijke voorstelling* van documenten.
Vijf stappen:

1. verwijderen van formatterings- en markup elementen
2. tokenization
3. filtratie
4. bepalen van stammen
5. wegen

De eerste stap wordt bij tekst-databanken nogal eens weggelaten. Op het web is deze stap belangrijk omdat documenten in allerlei formaten gecreëerd worden.

6.2.1 Document linearisatie

Document wordt omgevormd in een stroom van termen:

1. verwijderen van markup en formattering: bijv. bij een HTML document weglaten van alle tags en tekst binnen tags, inclusief scripts en commentaar lijnen
2. tokenization: omvormen naar kleine letters, verwijderen van leestekens, woordsplitsingen ongedaan maken

Resultaat:

1. een beschrijving van een coherente stroom van termen
2. deze tekststroom moet de bedoelde betekenis, thema, topics, subtopics, ... omvatten
3. de positie van de termen in deze tekststroom is bepaald door de formattering in de oorspronkelijke tekst

6.2.2 Filtratie

Beslissen welke termen zullen gebruikt worden om het document voor te stellen zo dat deze termen kunnen gebruikt worden voor

1. het beschrijven van de inhoud van het document
2. het onderscheiden van het document van andere documenten in de verzameling

Frequent voorkomende termen kunnen hiervoor niet gebruikt worden:

1. het aantal documenten dat relevant is voor een zoekvraag is waarschijnlijk slechts een klein deel uit de verzameling; een term die effectief relevante van niet-relevante documenten kan onderscheiden, is waarschijnlijk een term die in een beperkt aantal documenten voorkomt
2. termen die voorkomen in vele conteksten definiëren niet een topic of een subtopic van een document

Dus: verwijderen van stopwoorden

6.2.3 Bepalen van stammen

Reduceren van woorden zoals *computer*, *computing*, *compute* naar comput
of *walks*, *walking*, *walker* naar walk
Rudimentair: een heuristisch proces waarbij de eindletters van een term weggelaten worden in de hoop de stam over te houden.
Verschillende *stambepalende algoritmes* voor verschillende natuurlijke talen beschikbaar.

- documenten die een variatie van een gegeven zoekterm bevatten, worden in het zoekresultaat opgenomen
- reductie van de grootte van het geïnverteerd bestand
- maar, soms niet praktisch voor de eindgebruiker: gemakkelijker om de verschillende vormen in de vraag op te nemen.

Lemmatisatie: gesofistikeerder door gebruik te maken van een vocabularium en de morfologische analyse van de woorden
Bijvoorbeeld: *zagen* afkappen tot zag of omvormen tot zien of zaag afhankelijk van de analyse

6.2.4 Wegen van termen

Een gewicht toekennen aan elke term die de belangrijkheid aangeeft:
is een document met die term relevant voor een zoekproces met die term
Principe: een hoog gewicht toewijzen aan een term die frequent voorkomt in een document maar weinig in andere documenten
Berekening: gewicht $= tf \times idf$
term frequentie (tf) = frequentie van een term in een document
invers document frequentie (idf) $= \log \frac{\text{totaal aantal documenten in de verzameling}(N)}{\text{aantal documenten dat de term bevat}(df)}$
(tf) als een term veel keer voorkomt in een document, dan is de term waarschijnlijk betekenisvol om de inhoud van het document voor te stellen
(idf) hoe minder documenten de term bevatten, hoe beter de term is om documenten met de term te onderscheiden van documenten zonder de term

alternatief: het tf gewicht van een term reduceren met een factor die groeit met zijn verzameling freqeuntie (het aantal keer dat een term voorkomt in de volledige verzameling van documenten)

6.2.5 Indexing: voorbeeld

`<p><i>`Interactive`</i>`
query expansion modifies queries using terms from a user. `<i>`Automatic`<i>`
query expansion expands queries automatically. `</p>`

markup-vrije tekst

Interactive query expansion modifies queries using terms from a user. Automatic query expansion expands queries automatically.

tokenisation

interactive query expansion modifies queries using terms from a user automatic query expansion expands queries automatically

stopwoorden verlaten

interactive query expansion modifies queries terms automatic query expansion expands queries automatically

stammen maken

interact queri expan modifi queri term automat queri expan expand queri automat

termen wegen

automat	28	expand	28
expan	17	interact	28
modifi	17	queri	28
term	17		

6.3 Gelijkaardigheid en relevantie

De *gelijkaardigheid* van een document uit de totale verzameling en de vraag van de gebruiker wordt bepaald op basis van het verschil in de gewichten van de termen van het document en de gewichten van de termen in de vraag.
Een document is *relevant* voor een zoekvraag wanneer deze gelijkaardigheid groot is.
Gelijkaardigheid(document,zoekvraag)

$$= \sum_{\text{alle zoektermen}} \text{gewicht van de zoekterm} \times \text{gewicht van de term in het document}$$

Voor het gewicht van een term in het document kan TF*IDF gebruikt worden.
Een probleem hierbij is dat langere documenten meer gewicht krijgen, gewoon omdat ze meer woorden bevatten.
Dus, normalisatie van de gewichten van een document, bijv. cosinus normalisatie:

$$\frac{\sum_{\text{alle zoektermen}} \text{gewicht van zoekterm} \times \text{gewicht van term in document}}{\sqrt{\sum \text{gewicht van zoekterm}^2} \times \sqrt{\sum \text{gewicht van term in document}^2}}$$

Voorbeeld: PageRank van Google gebruikt de link structuur van het web als een indicator voor de waarde van een individuele webpagina A:

- als veel pagina's (T_i) een link hebben naar A, dan krijgt A meer gewicht
- als zo'n pagina met een link naar A zelf *belangrijk* is, dan weegt deze pagina meer op het gewicht van A (een belangrijke pagina met een link naar een andere pagina maakt deze pagina belangrijker)

$$PR(A) = (1-d) + d(\frac{PR(T_1)}{C(T_1)} + \cdots + \frac{PR(T_n)}{C(T_n)}) \quad \begin{cases} d \text{ dempingfactor} \in [0,1], (0.85) \\ C(T_i) \text{aantal links vertrekkend uit } T_i \end{cases}$$

Bij het meten van de gelijkaardigheid tussen een zoekvraag en een webpagina wordt gebruik gemaakt van tekst-matching technieken. Hierbij worden meer dan 100 factoren in rekening gebracht om te bepalen welke webpagina's het meest relevant zijn voor een zoekvraag:

- zoektermen in elkaars nabijheid in de pagina;
- zoektermen in dezelfde volgorde in de pagina als in de vraag;
- zoektermen in belangrijke onderdelen van de pagina (bijv. titel).

Evaluatie van de performantie van een zoekmachine. Het resultaat van een IR systeem is een gerangschikte lijst van documenten. De kwaliteit van dit resultaat hangt af van het feit of deze gerangschikte lijst van documenten *bevredigend* is voor de gebruiker.
Ten opzichte van een gegeven vraag kan de volledige ruimte van documenten opgedeeld worden in vier verzamelingen:

niet relevant voor gebruiker gevonden door systeem	relevant voor gebruiker gevonden door systeem
niet relevant voor gebruiker niet gevonden door systeem	relevant voor gebruiker niet gevonden door systeem

$$\begin{aligned}
\text{recall} &= \frac{\text{aantal gevonden relevante documenten}}{\text{totaal aantal relevante documenten}} \\
\text{precision} &= \frac{\text{aantal gevonden relevante documenten}}{\text{totaal aantal gevonden documenten}} \\
\text{fallout} &= \frac{\text{aantal gevonden niet-relevante documenten}}{\text{totaal aantal niet-relevante documenten}}
\end{aligned}$$

In waarschijnlijkheidstermen met A de relevante documenten, B de gevonden documenten en N totaal aantal documenten:

- recall: schatting van de voorwaardelijke waarschijnlijkheid dat een document gevonden wordt, gegeven dat het relevant is ($P(B|A)$);
- precision: schatting van de voorwaardelijke waarschijnlijkheid dat een document relevant is, gegeven dat het gevonden wordt ($P(A|B)$);
- fallout: schatting van de voorwaardelijke waarschijnlijkheid dat een document gevonden wordt, gegeven dat het niet relevant is ($P(B|\bar{A})$);

Tussen deze drie maatstaven bestaat een functioneel verband wanneer de generaliteitsparameter (G) in rekening gebracht wordt:

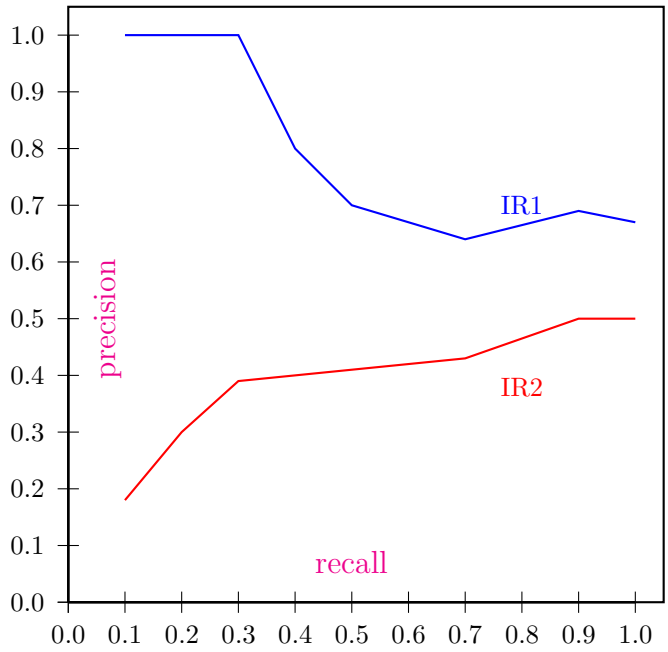
$$P = \frac{R \times G}{(R \times G) + F(1-G)} \quad \text{met} \quad G = \frac{\text{totaal aantal relevante documenten}}{\text{totaal aantal documenten}}$$

Deze generaliteitsparameter is een maatstaf voor de dichtheid van de relevante documenten in de volledige verzameling van documenten.
Probleem: totaal aantal relevante document niet gekend
gouden standaard waarbij alle documenten beoordeeld worden als relevant of irrelevant voor elke zoekvraag, wordt gewoonlijk manueel geconstrueerd door evaluators
In een normale verzameling documenten is het onmogelijk om elk document ten opzichte van elke zoekvraag te evalueren. Daarom wordt een *gepoolde relevantie inschatting* uitgevoerd waarbij meerdere zoekmachines in gezet worden die elk een aantal documenten voor elke zoekvraag opvragen.

Meestal is er een trade-off tussen recall en precision: hoe meer documenten er gevonden worden, hoe meer irrelevante documenten hierbij zijn.

Combinatie van recall en precision:

- F maatstaf:
$$F = \frac{2 \times recall \times precision}{recall + precision}$$
- 11-point average precision



11 intervallen voor recall tussen 0.0 en 1.0

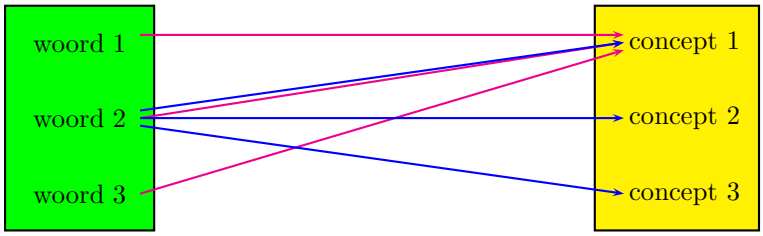
voor elke recall waarde: gemiddelde waarde voor precision

in het voorbeeld is systeem IR1 beter dan IR2:

voor elk recall niveau heeft IR1 een hogere precision waarde

Verklaring: natuurlijke taal

1. zoektermen en/of documenten: verschillende talen
2. ook wanneer alles in eenzelfde taal weergegeven is:
   - een woord is niet gelijk aan een concept
   - een concept is niet gelijk aan een woord
   - relaties tussen concepten (bijv. *computers* / *architectuur*)



- Een concept kan niet beschreven worden door slechts één woord of term. En dat is een mogelijke oorzaak van een lage recall van het zoekproces
- Betekenis van vele woorden is dubbelzinnig: een mogelijke oorzaak van een lage precision van het zoekproces.

Verhogen van performantie

Time flies like an arrow
Fruit flies like a banana

Time flies like an arrow
Fruit flies like a banana

Time flies like an arrow
Fruit flies like a banana

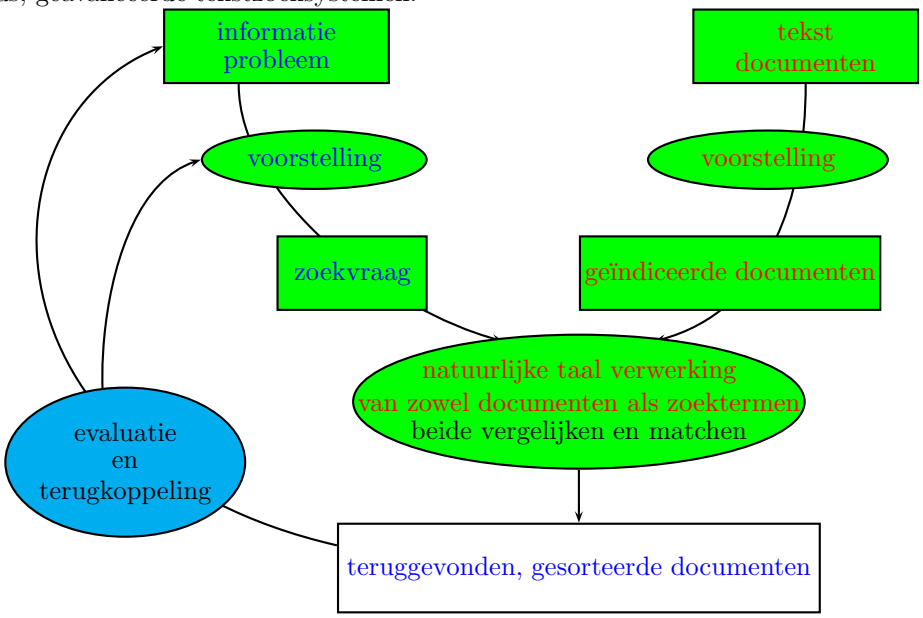
recall

- zoekvraag uitbreiden: meer termen om het concept te omvatten: synoniemen, specifiekere termen, spellingvarianten

precision

- dubbelzinnigheid uit de zoekvraag uithalen
- classificatie codes toevoegen aan elk gestockeerd document en deze codes meedelen aan de gebruiker

Dus, geavanceerde tekstzoeksystemen:



6.4 Oracle Text

De databank bevat tabellen met velden met lange teksten. Zoeken op waarden evolueert naar complexe zoekvragen in tekst waarbij meerdere termen, gewogen termen en score-rangschikking belangrijk zijn.
In tekst zoeken is verschillend van in data zoeken:

- woorden hebben verschillende betekenissen
- verwantschap met andere woorden
- zoeken naar woorden die dicht bij elkaar staan in de tekst

Hiervoor is meer nodig dan de standaard relationele operatoren: Oracle Text: uitbreiding van SQL met tekstindices

Tekst in een databank. In een tabel kan tekst op verschillende manieren opgenomen worden:
* veld met als waarde de bestandsnaam
* veld van type varchar2 : maximaal 4000 tekens;
* veld van type clob : meer dan 4000 tekens;
* veld van type blob : voor binaire informatie, bijv. beelden

```
create table boek
(
      bnr     integer primary key,
      titel   char(20),
      auteur  char(20),
      datum   date,
      tekst   varchar2(1000)
);
insert into boek values (1,'fruit eten', 'mie groen', '24/10/2008',
```

```
'Het is zeer belangrijk om veel fruit te eten. Appelen en peren
zijn de meest gekende inlandse soorten. Daarnaast is er ook heel
wat uitheems fruit, bijvoorbeeld bananen.
Ook apen en beren lusten sommige fruitsoorten.');

insert into boek values (2,'groenten', 'jos gezond', '26/12/2008',
'Groenten zijn onontbeerlijk. Prei, tomaten en selder kunnen gemakkelijk
via soep geinhaleerd worden. Bloemkolen, sla en witloof zijn niet zo lekker
voor de soep. Maar met een aangepast sausje en wat appelen of peren
geven deze groenten veel mogelijkheden bij aardappelen.
Een bijzondere soort zijn de kruiden: thijm, bieslook, dille en laurier.');
```

6.4.1 Indexen

Verschillende soorten: CONTEXT en CTXCAT tekstindices.

```
create index boek_content on boek(tekst)
indextype is ctxsys.context;

create index boek_ctxcat on boek(tekst)
indextype is ctxsys.ctxcat;
```

Voordeel: synchronisatie van de gegevens in de basistabel en in de tekstindexen

Op basis van zo'n tekstindex: mogelijkheden van tekst zoeken sterk uitgebreid:
sleutelwoord contains (context index) en catsearch (ctxcat index) in de where van een select
Na aanpassingen in de basistabel, moet een content index hersynchroniseerd worden:

```
begin    ctx_ddl.sync_index('boek_content');    end;
```

6.4.2 Scoring algoritme

Bij de berekening van de relevantie score van een gevonden document wordt een inverse frequentie algoritme gebruikt (formule van Salton):

- hierbij wordt verondersteld dat frequent voorkomende termen in een document *ruis*termen zijn en dus lager scoren;
- voor een hoge score moet de zoekterm frequent voorkomen in het document, maar weinig in de volledige documenten verzameling.

De score berekening is gebaseerd op een tabel waarin in functie van het aantal documenten in de documentenverzameling (N) aangegeven wordt hoeveel keer (T) de term moet voorkomen om een score van 100 te halen; hierbij wordt verondersteld dat de term slechts in één document in de verzameling voorkomt

N	T
1	34
5	20
10	17
50	13
100	12
500	10
1000	9
10000	7
100000	5
1000000	4

Voorbeeld. Een verzameling van 5000 documenten, waarbij de term processor tenminste eenmaal in elk document voorkomt. Er is één document (D) met vijf maal de term processor en vijf maal de term CUDA. Geen enkel ander document bevat de term CUDA. Alhoewel de frequenties van de twee termen in document D dezelfde zijn, krijgt CUDA een hogere score dan processor:

- processor komt frequent voor in de verschillende documenten van de set

- CUDA : zeer infrequent in de volledige documentenverzameling.

Zelfs als de frequentie van CUDA slechts 4 zou zijn, en die van processor 5, dan nog zou de score voor CUDA hoger zijn.

- Documenten toevoegen aan de verzameling die de term CUDA bevatten, zal de score van deze term in document D doen dalen.
- Documenten toevoegen aan de verzameling die de term CUDA niet bevatten, zal de score van deze term in document D doen stijgen.

6.4.3 Zoeken

Exacte match: één woord : zoeken naar titels waarbij de corresponderende tekst het woord "bananen" bevat.
Op basis van de context index:

```
select titel, score(5) from boek
where contains(tekst, 'bananen', 5) > 0;
```

Tijdens het zoeken wordt voor elke rij uit de boek tabel een score berekend. Indien deze score groter is dan nul, dan is de voorwaarde van de where waar. In dit voorbeeld is gekozen voor een drempelwaarde gelijk aan nul; maar dit kan ook een andere getalwaarde zijn.
De berekende score wordt ook in het resultaat getoond dmv. de score functie. Deze score functie kan ook in de group by en in de order by component gebruikt worden.

Op basis van de ctxcat index: (interne score kan niet getoond worden)

```
select titel from boek
where catsearch(tekst, 'bananen', 5) > 0;
```

Exacte match: meerdere woorden : teksten waarin zowel het woord "bananen" als het woord "peren" voorkomen:

```
select titel, score(5) from boek
where contains(tekst, 'bananen and peren', 5) > 0;
```

De berekende scores voor het woord "bananen" en het woord "peren" moeten beide groter zijn dan de drempelwaarde. In plaats van het sleutelwoord and kan ook het symbool `&` gebruikt worden. (In SQL*Plus moet dan wel de specifieke betekenis uitgeschakeld worden.)

Teksten waarin de woorden "bananen" of "peren" voorkomen:

```
select titel, score(5) from boek
where contains(tekst, 'bananen or peren', 5) > 0;
```

Als één van de twee woorden een score oplevert die groter is dan de drempelwaarde, is aan de where voorwaarde voldaan. Naast het sleutelwoord or kan ook het symbool `|` gebruikt worden.
Een accum lijkt op een or, maar heeft bovendien als effect dat een document dat beide zoektermen bevat (bv. zowel "appelen" als "peren") hoger zal scoren dan een document dat maar één van beide bevat:

```
select titel, score(5) from boek
where contains(tekst, 'bananen accum peren', 5) > 0;
```

Verminderen van de score van de eerste term met de score van de tweede term:

```
select titel, score(5) from boek
where contains(tekst, 'bananen minus peren', 5) > 0;
```

Een alternatief voor het minus sleutelwoord is het - symbool.

Om een rij (of meerdere rijen buiten beschouwing te laten kan het not sleutelwoord (of het ~ symbool) gebruikt worden.

```
select titel, score(5) from boek
where contains(tekst, 'bananen not peren', 5) > 0;
```

Meerdere woorden . Wanneer de where voorwaarde complexer wordt met meerdere woorden en operatoren kunnen best haakjes gebruikt worden:

```
select titel, score(5) from boek
where contains(tekst, 'fruit or (bananen and peren)', 5) > 0;

select titel, score(5) from boek
where contains(tekst, '(fruit or bananen) and peren', 5) > 0;
```

Bij meerdere zoektermen kunnen bij de verschillende termen verschillende gewichten toegekend worden (gewogen score):

```
select titel, score(5) from boek
where contains(tekst, 'bananen*2 or peren*1', 5) > 0;
```

Meerdere woorden als een deel van een zin:

```
select titel, score(5) from boek
where contains(tekst, 'appels {and}  pears', 5) > 0;
```

Meerdere woorden in elkaars nabijheid:

```
select titel, score(5) from boek
where contains(tekst, 'bananen near peren', 5) > 0;
```

Hoe dichter de woorden bij elkaar staan, hoe hoger de score.

Bijkomende voorwaarde: met een maximum aan tussenliggende woorden:

```
select titel, score(5) from boek
where contains(tekst, 'near((bananen,peren),10)', 5) > 0;
```

Meerdere woorden met zelfde volgorde in document als bij zoektermen:

```
select titel, score(5) from boek
where contains(tekst, 'near((bananen,peren),10,TRUE)', 5) > 0;
```

Geen exacte match .

- met wildcards:

  ```
  select titel, score(5) from boek
  where contains(tekst, 'banan% per__', 5) > 0;
  ```

  % staat voor één of meerdere tekens; _ voor één teken

- met dezelfde stam: handig wanneer men niet de exacte verbuigings- of vervoegingsvorm kent zoals die voorkomt in de te doorzoeken teksten; stamuitbreidingen van het engelstalige woordje *play* zijn bijvoorbeeld `plays`, `playing`, `played`, `playful`; stamuitbreidingen worden aangegeven door de stam te laten voorafgaan door het dollar teken:

  ```
  select titel, score(5) from boek
  where contains(tekst, '$eet', 5) > 0;
  ```

- fuzzy match: het zoeken wordt uitgebreid naar woorden die gelijkaardig gespeld worden, maar niet noodzakelijk dezelfde woordstam hebben:

```
select titel, score(5) from boek
where contains(tekst, '?beren', 5) > 0;
```

  Op basis van de `?` operator zullen ook teksten waarin het woord "peren" voorkomt, een score groter dan nul geven.

- gelijkaardig klinkend: de zoekterm wordt voorafgegaan door het `!` symbool

```
select titel, score(5) from boek
where contains(tekst, '!grate', 5) > 0;
```

  Dit is gelijkaardig aan de soundex functie in SQL.

ABOUT: zoeken naar thema's van (engelstalige) documenten:

```
select titel, score(5) from boek
where contains(tekst, 'about(fruit)', 5) > 0;
```

Er wordt gezocht naar termen die te maken hebben met het thema van het document in plaats van naar specifieke termen in het document.

Alternatieve score : aantal maal dat zoekwoord in een document voorkomt:

```
select titel, score(5) from boek
where contains(tekst, '<query>
          <textquery>bananen</textquery>
          <score datatype="INTEGER" algorithm="COUNT" />
                    </query>', 5) > 0;
```

Ook mogelijk `datatype="FLOAT"`: geeft een nauwkeuriger default score

A Oracle SQL*Plus

Basis werking. Starten met `% sqlplus /`

Na de prompt kunnen SQL statements ingetikt worden en ook *SQL*Plus* commands:

- `EXIT` of `QUIT`: stoppen.
- `DESCribe <tabelnaam>`: lijst van tabel met veldnamen en types.
- `DESCribe <functienaam>`: functie met returnwaarde en lijst van argumenten.
- `GET <filenaam>`: lees de query uit het genoemde bestand in de query buffer.
- `EDit`: editeer de actuele query buffer.
- `RUN` of `/`: stuur de actuele query buffer naar de backend, om uit te voeren.
- `SAVe <filenaam>`: schrijf de actuele query buffer weg naar het genoemde bestand.
- `List`: druk de inhoud van de query buffer af.
- `@<filenaam>` of `STArt <filenaam>`: lees en voer de queries uit het genoemde bestand uit.
- `SHOW USER`: toon de Oracle gebruikersnaam.
- `SHOW ALL`: toon de waarde van elke parameter in SQL*PLUS.
- `SHOW ERRORS`: toon de fouten na een "compilation with errors"van een PL/SQL blok.
- `HOST <unix_command>` (of `!<unix_command>`): uitvoeren van unix_command.

Datumfuncties:

```
SQL> select sysdate from dual;
SQL> select to_char(sysdate, 'MM') from dual;
SQL> select to_char(sysdate, 'IW') from dual;
SQL> select to_char(sysdate, 'YYYY') from dual;
```

Data dictionary:

```
SQL> select table_name from user_tables;
SQL> select view_name from user_views;
SQL> select table_name,column_name,date_type from user_tab_columns;
SQL> select table_name,search_condition from user_constraints;
```

Importeren van gegevens: `% sqlldr / control=meubel.ctl`

Formaatspecificatie:

9	tonen van significante cijfers	
0	markeren van de plaats vanaf waar leidende nullen getoond worden	
A<getal>	voor characters, *getal* geeft de kolombreedte aan	
TO_CHAR(datum, formaat)	D	dag van de week
	DD	dag van de maand
	DY	afgekorte dagnaam
	IW	week van het jaar
	MM	maandnummer
	MON	afgekorte maandnaam
	YYYY	jaartal
	YY	laatste twee cijfers van het jaartal

Merk op. Om het leven wat gemakkelijker te maken:

`% sql` en `% ladendb iets.ctl`

B Een databasetoepassing

B.1 Omschrijving van de gegevens

In de laatste faze van de ontwikkeling van een nieuw geneesmiddel moet dat middel uitgetest worden op een groot aantal mensen. Een niet nader genoemd farmaceutisch bedrijf beschikt hiervoor over een lijst van personen die aan zo'n test kunnen deelnemen. Van al deze personen wordt geslacht (**pgs**), al of niet roker (**pro**) zijn, naam (**pnm**), voornaam (**pvn**), postcode (**ppc**), woonplaats(**pwp**) en geboortedatum(**pgb**), bijgehouden. Er bestaat een lijst van middelen die in testfaze zijn. Attributen hier zijn naam (**naam**), soort (**soort**) (creme, pil, ...), kleur (**kleur**) en actief bestanddeel (**actief**). Voor het uittesten van een bepaald geneesmiddel wordt een test opgezet. Hiervan wordt de begindatum (**tbegin**) en einddatum (**teind**) bijgehouden. Naast het geneesmiddel is er ook een placebo (met attributen naam, soort en kleur) bij de test betrokken. Er moet ook aangegeven worden welke proefpersonen aan de test deelnemen, ofwel als persoon die effectief het geneesmiddel neemt of als controlepersoon die een placebo toegediend krijgt. Na afloop van de test wordt voor elke proefpersoon een resultaat (**res**) ingevuld dat aangeeft hoe de persoon op het geneesmiddel of placebo gereageerd heeft. Een proefpersoon kan in de loop van de tijd aan verschillende testen deelnemen.

Opgave 1. Maak een analyse van deze gegevens door middel van ER-modellering. Er kan best bij elke entiteit een identificatienummer (**Xnr** met X de eerste letter van de omvattende entiteit) als eerste attribuut toegevoegd worden.
Deze voorbereiding moet afgegeven worden in de eerste practicum zitting.

Opgave 2.

1. Maak een voorstel van een starschema voor een data warehouse met informatie over het totaal aantal deelnemers en het aantal positief reagerende deelnemers van de verschillende testen, over de verschillende geneesmiddelen.
2. Bepaal zelf welke dimensies hierbij eventueel van belang kunnen zijn.

Deze voorbereiding moet afgegeven worden in de zesde practicum zitting.

B.2 Opgaven voor de practicumzittingen

B.2.1 Creatie

1. Creëer de verschillende tabellen met behulp van SQL-statements.
2. Schrijf het *Insert* statement van uw eigen gegevens voor de tabel met de gegevens van de proefpersonen.

B.2.2 Eenvoudige SQL statements

Elk statement moet bewaard worden in een apart bestandje waarvan de naam met de letter **e** begint en gevolgd wordt door de nummer van de opgave. (Deze nummer moet uit twee cijfers bestaan, eventueel moet dus een `0` tussengevoegd worden.) Gebruik hiervoor het commando `SAV <naam>`.

1. Geef een overzicht van de testen.
2. Geef een overzicht van al de middelen.
3. Geef de nummers en de namen van alle middelen.
4. Geef de testnummers en begindata van de testen.
5. Geef de nummers, de namen en de voornamen van de vrouwelijke personen.
6. Geef de namen van de middelen die van het soort pil zijn.
7. Geef de lijst van testen die actueel nog lopen.

8. Geef de actuele datum en hernoem dit veld naar vandaag.
9. Geef de namen van alle middelen die eindigen op 'us'.
10. Geef de voornamen van alle personen die eindigen op 'ef'.
11. Geef een lijst van personen die geboren zijn tussen 1959 en 1960.
12. Geef een lijst van middelen met nummer tussen 100 en 150.
13. Geef een lijst van alle rokers uit Schriek.
14. Geef een lijst van alle groene poeders.
15. Geef een lijst van middelen met blauwe, groene of witte kleur.
16. Geef een lijst van middelen die ofwel pil of poeder of siroop zijn.
17. Geef het minimum, gemiddelde en maximum van de resultaten van de middelen met identificatienummer kleiner dan 200 over alle deelnames.
18. Geef de leeftijd van de jongste en oudste persoon en ook de gemiddelde leeftijd van alle personen. Hernoem uw kolommen zinvol.
19. Geef een lijst van namen van personen met leeftijd in jaren gesorteerd op leeftijd van oud naar jong.
20. Geef alle info van de testen gesorteerd op placebonummer van groot naar klein.
21. Geef een overzicht van de afgeronde testen miv duurtijd gesorteerd op duurtijd.
22. Geef een lijst van deelnames waarvan het middelnummer kleiner is dan 200 gesorteerd op resultaat.
23. Geef de totale som van de resultaten.
24. Geef het aantal deelnames waarvan het middelnummer groter is dan 200.

B.2.3 Groeperen

Elk statement wordt bewaard in een apart bestandje waarvan de naam met de letter **g** begint en gevolgd wordt door de nummer van de opgave.

1. Geef een lijst met nummers van alle geneesmiddelen en bijhorend actief bestanddeel en alle placebo's.
2. Geef het aantal uitgevoerde testen per placebo.
3. Geef een lijst per geboortejaar met het aantal rokers.
4. Geef een geordende lijst van testnrs met per test aantal testpersonen.
5. Geef een lijst met het aantal mannen en vrouwen onderverdeeld in rokers en niet-rokers.
6. Maak een gesorteerde frequentietabel van voornamen van vrouwen, toon alleen deze met een frequentie groter dan 7.
7. Geef de som van de resultaten per testnummer geordend op testnummer.
8. Geef de som van de resultaten per middel geordend op middel.
9. Geef de som van de resultaten per ternummer en middel geordend op testnummer en middel.

B.2.4 Join van tabellen

Elk statement wordt bewaard in een apart bestandje waarvan de naam met de letter **j** begint en gevolgd wordt door de nummer van de opgave.

1. Geef een lijst met testen met actief bestanddeel van het bijhorende geneesmiddel.
2. Geef een lijst met lopende testen met naam van het bijhorende placebo.
3. Geef een lijst met testen met zowel de naam en kleur van het bijhorende geneesmiddel gesorteerd op naam van het geneesmiddel.
4. Geef een lijst met testen met naam van het bijhorende geneesmiddel en het totaal aantal betrokken personen (zowel die geneesmiddel als die placebo toegediend gekregen hebben).

5. Geef een lijst met testen met naam van het bijhorende geneesmiddel en het aantal betrokken personen die het geneesmiddel toegediend gekregen hebben.
6. Geef een lijst met *langlopende* testen met naam van het bijhorende geneesmiddel en het aantal betrokken personen die het geneesmiddel toegediend gekregen hebben (een langlopende test is een test met een duurtijd van meer dan 50 dagen).
7. Geef de namen van de personen die aan vijf testen deelgenomen hebben.
8. Geef een lijst met namen van personen met aantal testdeelnames in dalende volgorde.
9. Geef een lijst met namen van personen met aantal testdeelnames in dalende volgorde met inbegrip van de personen die aan geen enkele test deelgenomen hebben.
10. Geef een lijst van testen met gemiddelde leeftijd van deelnemers bij begin van de test.

B.2.5 Subqueries

Elk statement wordt bewaard in een apart bestandje waarvan de naam met de letter **s** begint en gevolgd wordt door de nummer van de opgave.

1. Geef naam en voornaam van oudste persoon.
2. Geef de langst durende test en deze duurtijd.
3. Geef de lijst met alle informatie van testen waarbij een blauw geneesmiddel getest wordt.
4. Geef de lijst met alle informatie van alle testen met deelnemers uit Limburg (ppc tussen 3000 en 3900).
5. Geef de lijst met alle informatie van alle geneesmiddelen die actueel getest worden.
6. Geef de namen van personen die aan een test deelgenomen hebben (mbv. EXISTS).
7. Geef de namen van personen die aan geen test deelgenomen hebben (mbv. EXISTS).
8. Geef het testnummer en aantal personen van de test waaraan het meeste aantal personen deelgenomen hebben (mbv. >=ALL).
9. Geef de geordende lijst van testen met per test het aantal testpersonen die het geneesmiddel nemen.
10. Geef de lijst van testen geordend op het gemiddeld resultaat voor het geneesmiddel groter dan 70.0 met per test dit gemiddeld resultaat.
11. Geef de personen met de vier beste resultaten van test 1.
12. Geef de personen met de vier beste resultaten van test 1 beperkt tot het echte geneesmiddel.

B.2.6 OLAP

Elk statement wordt bewaard in een apart bestandje waarvan de naam met de letter **w** begint en gevolgd wordt door de nummer van de opgave.

1. Geef het totaal aantal personen dat aan één of andere test deelgenomen heeft met daarbij ook het totaal aantal dat positief gereageerd heeft.
2. Geef per geneesmiddel het aantal mensen dat deelgenomen heeft met daarbij het aantal dat positief gereageerd heeft.
3. Combineer de twee vorige opgaven in één OLAP SQL statement.
4. Geef per geneesmiddel en per type proefpersoon de sommen. Met som wordt bedoeld *het totaal aantal mensen dat deelgenomen heeft en het totaal aantal dat positief gereageerd heeft.*
5. Geef per geneesmiddel het totaal aantal mensen dat deelgenomen heeft met daarbij het totaal aantal dat positief gereageerd heeft, en geef dit ook per type proefpersoon en ook de globale totalen.
6. Geef de sommen per geneesmiddel en per type persoon, de sommen per geneesmiddel en de totale sommen.
7. Geef de sommen per type persoon, per leeftijd en per geneesmiddel en ook alle geaggregeerde sommen.

8. Geef de sommen per type persoon, per leeftijdscategorie en per geneesmiddel en ook alle geaggregeerde sommen.
9. Geef de sommen per type persoon en per leeftijdscategorie voor het eerste geneesmiddel en ook alle geaggregeerde sommen.
10. Geef de sommen per provincie, per type persoon, per leeftijdscategorie en ook alle geaggregeerde sommen.

B.2.7 Information retrieval

Elk statement wordt bewaard in een apart bestandje waarvan de naam met de letter **t** begint en gevolgd wordt door de nummer van de opgave.

1. Geef alle gegevens van de `tekst`-tabel.
2. Geef tnr,titel en score > 0 voor het woord middel in de inhoud.
3. Geef tnr,titel en score > 4 voor het woord middel in de inhoud.
4. Geef tnr,titel en score > 0 voor het woord pijn in de inhoud.
5. Geef tnr,titel en score > 0 voor de woorden middel en pijn in de inhoud.
6. Geef tnr,titel en score > 0 voor de woorden middel of pijn in de inhoud.
7. Geef tnr,titel en de geaccumuleerde score > 0 voor de woorden middel en pijn in de inhoud.
8. Geef tnr,titel en score > 0 voor de woorden middel verminderd met de score voor pijn in de inhoud.
9. Geef tnr,titel en score > 0 voor de woorden pijn verminderd met de score voor middel in de inhoud.
10. Geef tnr,titel en score > 0 voor de woorden middel zonder pijn in de inhoud.
11. Geef tnr,titel en score > 0 voor de woorden middel verminderd met de score voor pijn waarbij je pijn $2\times$ meer gewicht geeft in de inhoud.
12. Geef tnr,titel en score > 0 voor de zin “veel soorten pijn” in de inhoud.
13. Geef tnr,titel en score > 0 voor de woorden pijn en lichaam in elkaars buurt in de inhoud.
14. Geef tnr,titel en score > 0 voor de woorden pijn en alarmsignaal in elkaars buurt in de inhoud.
15. Geef tnr,titel en score > 0 voor de woorden die beginnen met “mi” in de inhoud.
16. Geef tnr,titel en score > 0 voor het woord pijn met als extra voorwaarde dat de auteur Ramon Yves is, in de inhoud.
17. Geef tnr,titel en score > 0 voor het woord hartinfarct met als extra voorwaarde dat de titel “acetylsalicylzuur met uranus” is, in de inhoud geordend op datum.
18. Geef tnr,titel en score > 0 voor de woorden hoesten en keel met een maximum van 15 tussenliggende woorden in de inhoud.
19. Zelfde vraag als vorige maar de volgorde van de woorden in uw vraag moet zelfde zijn als die in de inhoud.
20. Tel het aantal keren dat het woord pijn in de inhoud voorkomt.

B.2.8 Extract van de inhoud van de tabellen

Tabel **persoon**:

PNR	PGS	PRO	PNM	PVN	PPC	PWP	PGB
323	v	j	Rao	Ann	9120	Melsele	16/02/78
324	m	n	Ramon	Yves	2800	Mechelen	23/02/61
325	m	j	Mattys	Bart	2221	Booischot	08/10/79
326	m	j	Petit	Joris	2360	Oud-Turnhout	03/08/65
327	m	j	Slootmaeckers	Nico	2220	Heist-op-den-Berg	02/05/63
328	v	n	Van Beers	Ilse	2221	Booischot	22/08/65
329	m	n	Koijen	Michiel	2860	Sint-Katelijne-Waver	14/06/61
330	m	n	Van Eyken	Nico	2140	Borgerhout	08/11/77
331	m	n	Meulenbergs	Sander	3090	Overijse	15/03/63
3236	v	j	Verlooy	Victoria	2811	Leest	26/04/60

Tabel **middel**:

MNR	NAAM	SOORT	KLEUR
101	lysanx	pil	blauw
102	flucma	pil	blauw
103	broncho	siroop	rood
104	dafalgan	pil	blauw
105	aspro	pil	wit
106	cilaza	pil	wit
107	seretide	poeder	groen
108	daktacor	creme	geel
201	helena	pil	blauw
202	venus	creme	geel
203	mercedes	siroop	rood
204	neptunus	poeder	wit
205	uranus	pil	wit

Tabel **geneesmid**:

GNR	ACTIEF
101	prazepamum
102	fluconazol
103	codeine
104	paracetamol
105	acetylsalicylicum
106	cilazapril
107	salmeterol
108	micanazoli

Tabel **placebo**:

GNR
201
202
203
204
205

Tabel **test**:

TNR	TBEGIN	EIND	TGENMID	TPLBMID
1	26/03/04	26/06/04	101	201
2	15/03/06	15/09/06	102	201
3	16/08/06	14/03/07	103	203
4	6/07/07	18/08/07	104	201
5	8/10/08	17/11/08	105	205
6	26/10/09	18/12/09	108	202
7	1/08/10	31/12/10	105	205
8	1/09/10	31/12/10	107	204

Tabel **deelname**:

PNR	TNR	MNR	RES
1082	4	104	86
983	4	104	79
581	4	104	71
172	4	201	31
2928	4	201	38
1394	4	201	55
2853	5	105	74
2680	5	105	78
1550	5	105	63
855	1	101	67
1869	1	101	85
2168	1	201	53
1975	2	102	73
2883	2	201	38
2394	3	103	68
2761	3	103	83
93	3	103	62