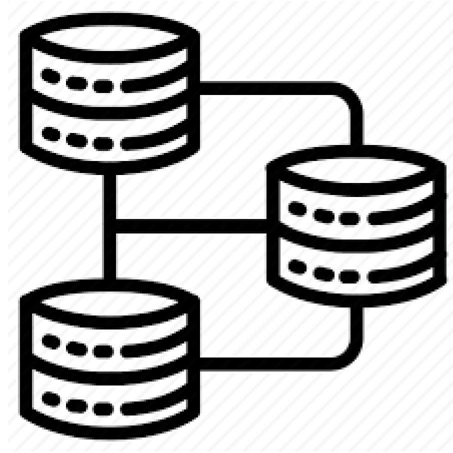


HET RELATIONELE MODEL EN NORMALISATIE

Johan Blomme
1997

www.johanblomme.net

INLEIDING

Omwille van de tactische en strategische mogelijkheden heeft de theoretische en praktische ontwikkeling van database marketing de laatste jaren een sterke vooruitgang gekend. Bedrijven onderkennen meer en meer dat kennis en informatie omtrent hun reële en potentiële markten essentieel worden in hun marketingbeleid, en competitieve voordelen verschaffen of kunnen verschaffen. De *database* wordt een noodzakelijk middel in hun marketingstrategie.

Parallel hiermee ziet men in de informaticasector een ontwikkeling van de technologie rond databaseconcepten. De kwantitatieve opslag van gegevens is hiervan een aspect ; andere aspecten hebben betrekking op het leggen van flexibele relaties tussen de opgeslagen gegevenselementen ten einde snelle en functionele interacties mogelijk te maken. Ook softwarepakketten voor beheer van marktgegevens zijn al veelvuldig op de markt. Recent doen ook gebruiksvriendelijke analysepakketten hun intrede. Zij maken het mogelijk om vrij eenvoudig adressenbestanden te analyseren, clusters van doelgroepen te ontdekken en aldus gefundeerder en gerichter beslissingen te nemen op het vlak van marketingcommunicatie.

De enorme groei van database marketing is geworteld in de filosofie om dicht bij de klant te staan, zijn of haar behoeften te kennen, in te vullen en hem of haar voortreffelijk te blijven behandelen na een verkoop. De groeiende mogelijkheden met betrekking tot automatisering en de toegenomen druk om marketing activiteiten op een efficiënte, degelijk gestructureerde en meetbare wijze uit te voeren, leidde tot een snelle ontwikkeling van *databasemarketing*.

Een informatiesysteem is een hulpmiddel, een tool om een bepaald proces te begeleiden. In het geval van een onderneming gaat het daarbij natuurlijk om een business-proces. De bedoeling van het gebruik van een systeem om zo'n proces te begeleiden is om het zo optimaal mogelijk te laten verlopen. Dit kan op twee manieren. Hetzij door de *efficiëntie* te verhogen, hetzij door de *effectiviteit* te verhogen. Het eerste is het domein van mensen in een operationele functie, het tweede dat van het management.

Traditioneel heeft men informatica ingezet om de operationele processen te begeleiden, het automatiseren van productie en administratie bijvoorbeeld. Waarschijnlijk omdat daar het meeste behoefte was aan optimalisatie en rationalisatie, maar even waarschijnlijk ook omdat het het gemakkelijkste was om te informatiseren. Het uitbouwen van systemen die de effectiviteit van de managementbeslissingen moeten ondersteunen, is iets wat de laatste vijf

(maximaal tien) jaar begonnen is, en dan nog maar in een minderheid van de bedrijven.

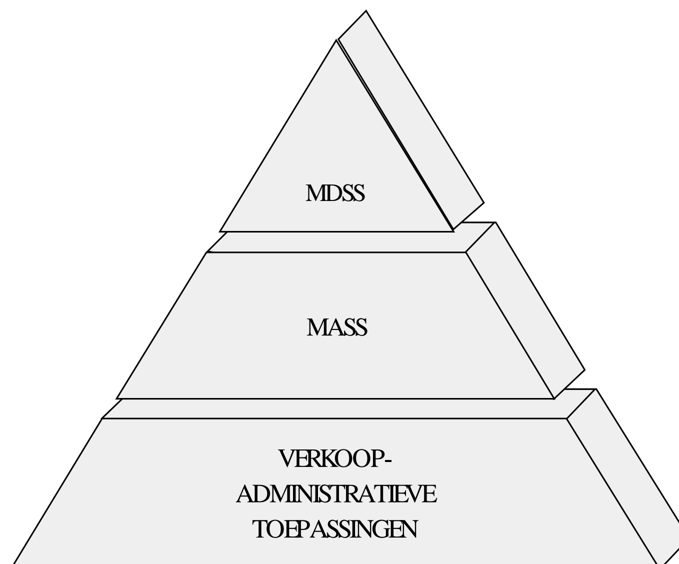
Indien we ons richten op wat er binnen een commercieel departement reilt en zeilt, dan vinden we enerzijds operationele processen, zoals het opbellen van mensen, het opnemen van bestellingen, het maken van facturen... Anderzijds vinden we er processen van tactische en strategische aard - over het algemeen 'beslissingen' genoemd. Hier denken we aan het nemen van een beslissing omtrent een investering, het kiezen van een doelgroep, het lanceren van een nieuw product. We zien dus twee totaal verschillende processen, andere mensen die erbij betrokken zijn, andere prioriteiten en dus ook andere systemen. In het eerste geval spreken we over *transactionele systemen* (OLTP : "On Line Transaction Processing"), in het tweede geval over *beslissingsondersteunende systemen* (OLAP : "On Line Analytical Processing").

1. Marketinginformatie-systemen (MIS)

In de eerste plaats zijn er de verkoopadministratieve systemen. Binnen de commerciële afdeling zijn er heel wat administratieve procedures : het beheren van de stock, het opnemen van bestellingen, het verzorgen van de leveringen, het factureren van de prestaties, het opvolgen van de betalingen. Deze procedures zijn vaak gestandaardiseerd, zeer repititief en overschrijden ook vaak de werking van het commerciële departement. Het financiële departement, productie en stockbeheer zijn andere diensten die erbij betrokken worden. Het een en ander heeft tot gevolg dat deze administratieve procedures al jaren zijn geautomatiseerd. Als we het tien jaar geleden hadden over het gebruik van informatica in business, dan hadden we het hierover. Vaak noemt men deze verkoopadministratieve systemen "mission critical".

Daarnaast zijn er enkele andere operationele taken binnen het commerciële departement. Meestal hebben die te maken met zaken die het verkoopmoment voorafgaan. Het screenen van "leads", het maken van offertes, het schrijven van afspraakbevestigingen, het plannen van promotionele acties, e.d. Maar ook met wat nà de verkoop komt : het geven van service, het behandelen van klachten, ... Eigenlijk hebben we het hier over de activiteiten die het marketingdepartement uitmaken. Communicatieverantwoordelijken en veldwerkers. Het gaat over marketingactie. Systemen die deze actie moeten ondersteunen, worden "marketing action support systems" genoemd.

Ten slotte zijn er de marketingbeslissingen die normalerwijze elke actie voorafgaan. Ook zij zijn van diverse aard. Van de strategische beslissing om te investeren met een bepaald product in een bepaalde markt (product/marktcombinaties), over het vastleggen van de jaarlijkse budgettten tot en met de concrete planning van een verkoper of het script van een telemarketeer. De beslissingen variëren dus sterk, maar de informatiebehoeften lijken sterk op elkaar. Telkens heeft men behoefte aan alle gegevens die een stukje van de puzzel kunnen helpen invullen, en aan een aantal modellen om al die gegevens bevattelijk en interpreteerbaar te maken. De systemen die daarbij moeten helpen, noemen we "marketing decision support systems".



Op basis van het voorgaande kunnen we tot de volgende definitie komen. Een marketinginformatie-systeem is de verzameling van alle informatiesystemen binnen het bedrijf, die als bedoeling hebben het commerciële departement te ondersteunen, zowel wat betreft het nemen van de beslissingen als wat betreft het uitvoeren ervan. Hierbij moeten we onthouden dat die diverse systemen (of toepassingen) in drie grote categorieën kunnen worden ondergebracht : de verkoopadministratieve toepassingen, "marketing action support systems" en "marketing decision support systems".

Nemen we de term 'databasemarketing' letterlijk, dan lijkt het sterk op een synoniem voor MIS. Marketing is een breed begrip, dat zowel slaat op het nemen van beslissingen als het uitvoeren van beslissingen betreffende de marketing van goederen, diensten en/of ideeën. Daarbij is een informatiesysteem van wezenlijk belang en een databank vormt een belangrijk onderdeel van een informatiesysteem.

Slaan we er evenwel de literatuur op na over het onderwerp, dan merken we dat men met de term 'databasemarketing' toch iets minder ambitieus is. Het blijkt te gaan om het voeren van marketingacties (dus operationeel) via nieuwe technieken (bijvoorbeeld gepersonaliseerd drukken) vanuit een klanten- en/of prospectenbestand. De nadruk ligt dus op die databank binnen het MIS, die sterk op klanten en prospects is afgestemd, en op het operationeel gebruik ervan. Het gaat dus om een aantal specifieke toepassingen (direct mail, telemarketing, gebruik van elektronische klantenkaart in een winkelpunt), die passen binnen het globale marketing-informatie-systeem.

Het MIS wordt, in de integrale versie, het hart van het commerciële departement. Het bevat immers alle informatie die de marketeers moet toelaten om hun toegevoegde waarde te realiseren en eventueel uit te breiden. Het ondersteunt iedereen binnen het commerciële departement (marketing en verkoop) bij het nemen van alle beslissingen op alle beleidsniveaus. Absolute noodzaak is daarbij wel het MIS te bekijken als één globaal systeem. Pas dan komen de voordelen van de inspanningen naar boven die moeten worden geleverd om bij elke actie ook de data te gaan stockeren. Pas dan wordt het ook mogelijk om experts op vlak van informatietechnologie te betrekken bij het uitbouwen van een systeem dat in ondersteuning van alle beslissingen en acties kan gaan werken. Pas dan wordt het mogelijk om het niveau van de losse, naast elkaar staande toepassingen te overstijgen.

Bij het overlopen van de diverse toepassingen binnen een MIS moeten we- zoals hoger reeds aangehaald - een onderscheid maken tussen transactionele toepassingen en toepassingen die een beslissingsondersteunende functie hebben.

1.1. Transactionele (data-inzameling) toepassingen

Transactionele toepassingen hebben als eerste functie het efficiënt laten verlopen van een bepaald proces, maar als belangrijke nevenfunctie het inzamelen van informatie. Bij de traditionele verkoopadministratieve systemen is dit een vanzelfsprekendheid. Bij verschillende toepassingen in de sfeer van MASS zien we echter dat de primaire bedoeling vaak data-inzameling is. De functie van verhoging van de efficiëntie mag echter nooit uit het oog verloren worden. Anders zal het systeem nooit geaccepteerd worden door de gebruikers.

Vaak worden verkoopadministratieve systemen niet beschouwd als een onderdeel van het MIS. Omdat ze administratief zijn ? Omdat ze traditioneel al jaren bestaan ? Omdat ze traditioneel het domein van EDP zijn ? Feit is wel dat in het kader van de optimalisatie van de kwaliteit van de service, marketing meer en meer intervenieert in de administratieve toepassingen. Daarnaast stellen we vast dat door de opkomst van verschillende alternatieve verkoopkanalen (vooral vanuit de hoek van direct marketing) ook vaak wijzigingen moeten worden doorgevoerd in de administratieve systemen. We denken hier bijvoorbeeld aan alles wat te maken heeft met het beheer van promotioneel materiaal, meer en meer complexe kortingsschema's, e.a.

1.2. Marketing Action Support Systems (MASS)

Deze toepassingen betreffen alle procedures binnen een commerciële afdeling buiten wat zich afspeelt tussen het bestel- en het facturatiemoment. In essentie gaat het hier om het beheren en opvolgen van alle contacten met alle mogelijke partijen binnen de commerciële context (relatiebeheer). Gemakkelijkheidshalve maken we een onderscheid tussen het zelf beheren van deze activiteiten en de organisatie van de administratie eromheen.

Ondanks de evolutie naar direct marketing blijft de rol van de verkoper nog steeds cruciaal. De contacten die de verkoper heeft met klanten en prospects zijn belangrijk en intensief. Ze moeten vooraf goed gepland worden en nadien goed gerapporteerd. Ze brengen een massa gegevens over de markt aan. Het beheren van deze contacten vanuit een klanten- en prospectendatabase vormt het onderwerp van een verkoopautomatiseringsproject. Zo'n project streeft ernaar alle relevante informatie aan een verkoper beschikbaar te stellen, en hem een instrument te bezorgen om zichzelf efficiënter te organiseren respectievelijk om te kunnen communiceren met al zijn collega's.

Functioneel sterk vergelijkbaar met de vorige toepassing, is telemarketing. Met een aantal functionaliteiten die afgestemd zijn op het typische karaker van telefonische contacten, zoals snelschermen en "scripting". Deze toepassingen kunnen natuurlijk ook gebruik maken van de diverse nieuwe mogelijkheden die de telecommunicatie-technologie biedt. In sommige sectoren waar de telefoon als contactmedium een zeer sterke vlucht heeft genomen, zullen we wel zeer specifieke toepassingen zien ontstaan.

Net als het individueel gesprek en het telefonische onderhoud is de brief een middel om contacten te leggen. Ook hier stellen we vast dat in de meeste gevallen het beheer van schriftelijke contacten past binnen het globale contact-managementsysteem, maar dat dit in sommige omgevingen het voorwerp uitmaakt van afzonderlijke toepassingen. In ieder geval moet het onderscheid gemaakt worden tussen individuele briefwisseling (prijsvoorstellen, bevestiging van afspraken, samenvattingen van bezoeken, ...) en groepsgeoriënteerde briefwisseling.

Door de steeds groter wordende druk, de drang naar efficiëntie en de complexer wordende marketingacties is het plannen van de diverse contacten cruciaal geworden. Het betreft hier zowel de planning van individuele agenda's en die van groepen, de opvolging van vergaderzalen, materiaal ... als de organisatie van de routings, e.a. Typische toepassingen die de laatste tijd meer en meer in opgang zijn, betreffen het volledige beheer van de promotionele activiteiten en de reclamecampagnes.

In de hele "contact cycle" is het moment van de offerte een van de belangrijkste. Het bepaalt in grote mate de kans tot succes en in vele gevallen neemt het maken ervan een groot deel in van de tijd van mensen in een commerciële functie. In sommige gevallen sluit het eerder aan bij de prospectie, in andere gevallen bij het gebeuren van "order processing".

1.3. Marketing Decision Support Systems (MDSS)

Aangezien het marketingproces een continu proces is van beslissingen nemen en beslissingen uitvoeren, stellen we vast dat data-inzameling en data-exploitatie elkaar voortdurend afwisselen. Men mag MASS en MDSS dan ook niet zien als twee totaal los van elkaar staande zaken. Wel kan een duidelijke lijn getrokken worden tussen het analyseren en rapporteren van informatie voor het ondersteunen van dagelijkse beslissingen en de meer tactische en strategische beslissingen.

In het eerste geval gaat het om meestal eenvoudige analyses en wordt er met informatie gewerkt op een niveau dat zeer nauw aansluit bij het niveau waarop de gegevens worden ingebracht. In het tweede geval zijn de modellen vaak complexer en moet de informatie naar meer geaggregeerde niveaus getrokken worden. Naast dit - toch wel belangrijke - onderscheid kan nog een opsplitsing gemaakt worden naar een aantal groepen van toepassingen.

We vertellen niets nieuws door te stellen dat managers weinig tijd hebben en dat hun belangrijkste taak niet ligt bij het verzamelen en voorbereiden van informatie. Daarom moet er gewerkt worden met *boordtabellen* : het ordenen en presenteren van de informatie op een dusdanige manier dat de manager onmiddellijk kan overgaan tot de interpretatie van de informatie zelf. Deze boordtabellen kunnen op papier of elektronisch worden aangeleverd. In het tweede geval gebruikt men de term “executive information system”.

Een specifieke term afkomstig uit een bepaalde technologische omgeving, maar die algemeen geworden is voor het aanduiden van het ophalen van gegevens uit een databank is “query reporting”. Meestal gaat het over het maken van lijsten en fiches die een synthetisch inzicht geven in bepaalde operationele fenomenen.

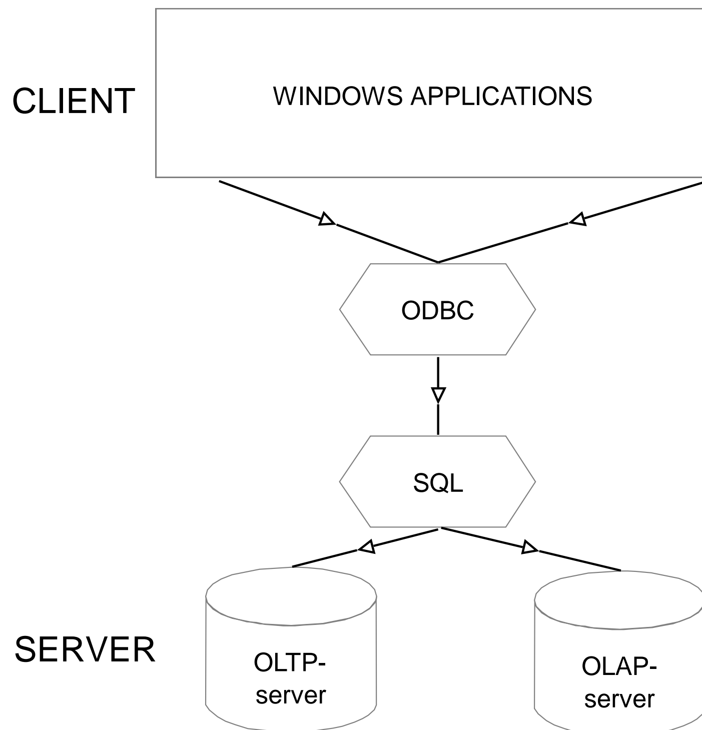
Naast de boordtabellen voor het topmanagement en de lijsten (“queries”) voor de mensen in een operationele functie bestaat er ook de behoefte om in sommige gevallen op zeer specifieke vragen een oplossing te vinden. Bijvoorbeeld bij de lancering van een nieuw product. Dit gebruik van informatie vinden we vooral terug bij businessanalisten, marktonderzoekers en in mindere mate ook bij productmanagers. Zij hebben minder behoefte aan voorgekauwde informatie, wel meer aan flexibiliteit. Voor dit gebruik van informatie zien we dat er aparte tools beschikbaar zijn, variërend van rekenbladen voor het eenvoudige werk tot specifieke statistische software (“data mining”)[1].

2. Architecturen voor multi-user databasetoepassingen

Als we het over informatiesystemen hebben, mogen we de technologie niet uit het oog verliezen. Vaak wordt als belangrijkste argument voor het opzetten van een MIS aangehaald dat de hoeveelheid gegevens die marketeers moeten beheersen, zo groot geworden is dat het niet meer kan zonder computer. Dat klopt, maar tegelijkertijd geeft het aan dat de informatisering niets toevoegt aan de vroegere manier van werken. En dat klopt niet. Vandaag biedt de informatisering mogelijkheden die de hele organisatie en werking van een commercieel departement kunnen wijzigen en verbeteren.

Een tweede opmerking betreft de verhouding tussen de commerciële gebruikers van de toepassingen en de ontwikkelaars ervan. Enkele jaren geleden - toen voor het eerst over de informatisering van het commerciële departement werd gesproken - bevonden we ons in een periode waar informatica voor het eerst in handen van niet-informatici terecht kon komen. De eerste marketing-informatiesystemen waren dan ook vaak de vrucht van het werk van marketeers. Dikwijls bestond er bij de professionele ontwikkelaars een zekere terughoudendheid om er hun medewerking aan te verlenen. Vandaag de dag is dit totaal veranderd. Het is inmiddels duidelijk dat - ondanks het blijkbaar eenvoudiger worden van de informaticatools - het ontwikkelen van toepassingen een aparte 'stiel' is. Er zijn nu voldoende producten op de markt waarmee het MIS kan worden opgebouwd. Het 'wiel opnieuw uitvinden' is niet langer nodig. Marketing is dus in de "mainstream" van informatica terechtgekomen, en dat is een goede zaak.

Het hart van het MIS is de databank. Aangezien het niet alleen gaat over het begeleiden van een transactioneel proces maar ook over het exploiteren van de gegevens voor het ondersteunen van diverse beslissingen, is de stockage van de gegevens uitermate belangrijk. Een aantal standaarden voor het opzetten van een MIS zijn : client/server, SQL en Windows[2].



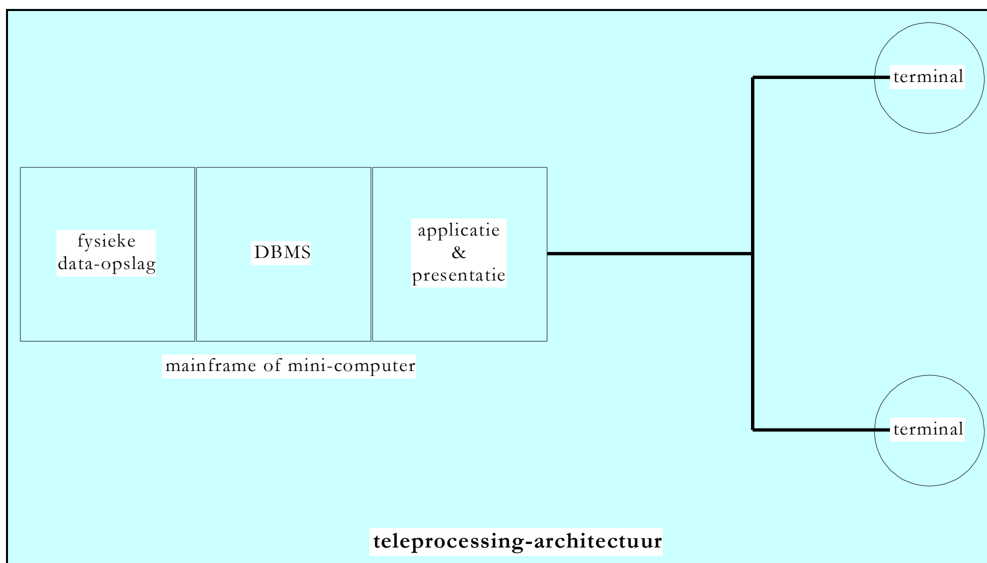
Na 1985 begonnen eindgebruikers hun afzonderlijke computers onderling te verbinden door middel van een zogeheten "local area network" (LAN). Met behulp van dergelijke netwerken kunnen computers veel sneller gegevens uitwisselen dan voordien mogelijk was. De eerste netwerken maakten vooral gebruik van gemeenschappelijke randapparatuur zoals grote schijfeenheden en printers. Verder kon men onderling communiceren via elektronische post. Op den duur wilden gebruikers ook gebruik maken van gemeenschappelijke databanken en dit leidde tot de ontwikkeling van *multi-user databasetoepassingen* voor lokale netwerken.

De multi-user architectuur voor LAN-databases verschilt aanmerkelijk van die voor mainframes en minicomputers. Op een mainframe of minicomputer wordt maar één processor gebruikt voor de databaseverwerking ; op LAN-systemen kunnen meerdere processoren tegelijkertijd bezig zijn met het verwerken van gegevens. Dit is een verbetering (er is meer verwerkingscapaciteit beschikbaar), maar het leidt ook tot meer complicaties (omdat de acties van de onafhankelijk werkende processoren moeten worden gecoördineerd). Om deze complicaties het hoofd te bieden, werd een nieuwe werkwijze ontwikkeld, de zogenaamde *client/server database-architectuur*. De client/server-architectuur vormt tegenwoordig de basis voor de meeste groepsdatabasetoepassingen.

2.1. Teleprocessing-systemen

Tegenwoordig hebben organisaties en bedrijven hun bedrijfsgegevens opgeslagen in een databasesysteem. Indien toegang tot de gegevens mogelijk is voor meerdere gebruikers tegelijk, is er sprake van een *multi-user databasesysteem*. Een databasesysteem laat zich het best beschrijven aan de hand van een *lagenmodel*. De lagen vormen samen de interface tussen de gebruiker en de database, en elke laag correspondeert met een verwerkingstaak van het database systeem. We onderscheiden vier lagen, nl. *opslag, management (beheer), verwerking en presentatie van de gegevens*.

Tot aan de jaren tachtig werd voor de realisatie van multi-user databasesystemen gebruik gemaakt van de mini/mainframe-architectuur. Deze monolitische architectuur bestaat uit *één centrale computer of processor met daaraan gekoppeld een netwerk van terminals*. Deze terminals beschikken niet over eigen 'intelligentie', anders dan nodig voor het afhandelen van scherm- en toetsenbordhandelingen. Alle vier genoemde verwerkingstaken vallen onder de verantwoordelijkheid van de centrale processor. Dat heeft een groot voordeel : centraal kan worden afgedwongen dat alle toegang tot de database verloopt via de managementlaag, *het databasemanagement-systeem (DBMS)*. Zo is het relatief eenvoudig om de kwaliteit van de database te bewaken. Maar omdat alle taken op één computer plaatsvinden, en elke taak een ander soort optimalisatie vereist, is het systeem als geheel moeilijk te optimaliseren.

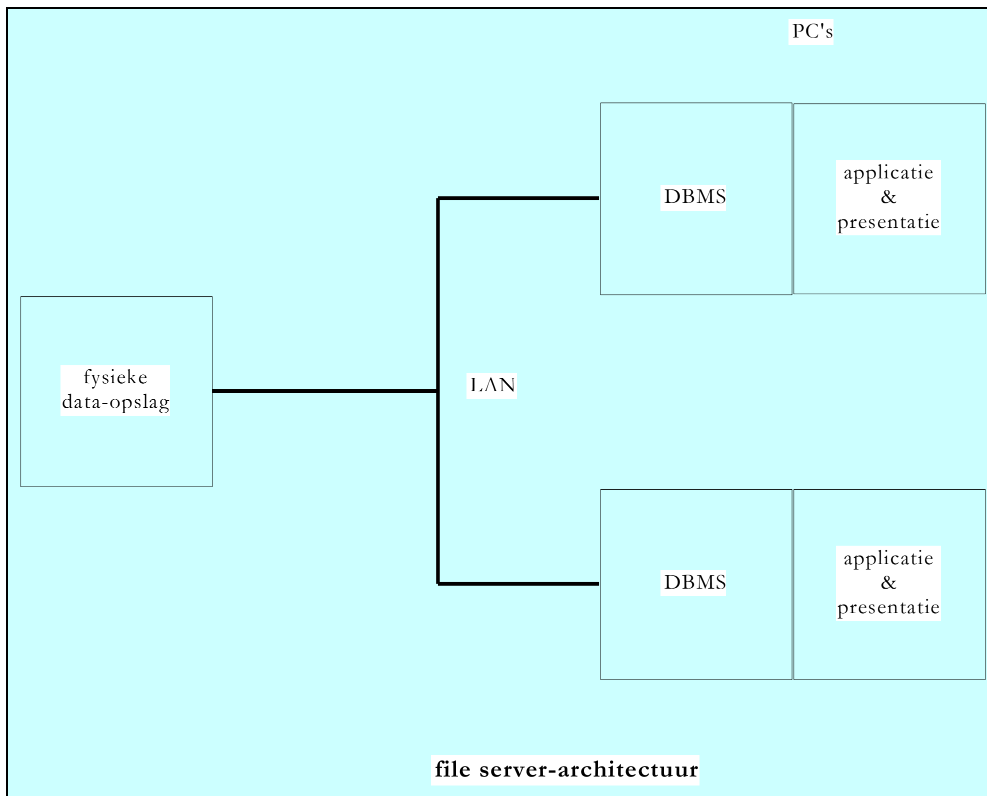


Het ondersteunen van een multi-user databasesysteem door gebruikmaking van één centrale computer die alle verwerking verzorgt, wordt *teleprocessing* genoemd (de verwerking gebeurt immers op een afstand). In een teleprocessing-systeem hebben de gebruikers de beschikking over terminals die transactieberichten versturen naar de centrale computer. Het communicatiedeel van het besturingssysteem van de centrale computer ontvangt deze berichten en stuurt ze door naar het juiste applicatieprogramma. Deze programma's roepen op hun beurt het DBMS aan en dit gebruikt het gegevensmanagementgedeelte van het besturingssysteem om de database te bewerken. Als een transactie gereed is, worden de resultaten geretourneerd naar de gebruiker via het communicatiegedeelte van het besturingssysteem.

Omdat zich aan de zijde van de gebruikers weinig intelligentie bevindt moeten ook alle commando's door de centrale computer verwerkt worden en moeten de resultaten ook over de communicatielijnen worden verstuurd. De gebruikers- interface is daarom meestal tekstgeoriënteerd (niet grafisch). Oorspronkelijk waren teleprocessing-systemen de gebruikelijke oplossing voor het ondersteunen van multi-user databasesystemen, maar met het dalen van de prijs-prestatieverhouding van computers en met name PC's kwamen andere architecturen meer in zwang.

2.2. "Resource sharing"-systemen

De afgelopen jaren is het gebruik van PC's enorm toegenomen. De personal computer biedt gebruikersvriendelijkheid en, nog belangrijker, eigen verantwoordelijkheid voor de gebruiker over de computer die op diens bureau staat. De PC's worden gebruikt voor administratieve toepassingen zoals tekstverwerking en spreadsheets, ook wel aangeduid als "office automation". Door het toegenomen persoonlijk gebruik ontstond er steeds meer behoefte om de ontstane gegevenseilanden met elkaar te koppelen. Hierdoor kwam het LAN op. Gebruikers konden bestanden met elkaar uitwisselen en een printer of andere randapparatuur met elkaar delen. Een veelgebruikte implementatie van een LAN bestaat uit een *file server* met daaraan gekoppeld een aantal PC's. De PC's draaien hun eigen besturingssysteem, aangevuld met software om de file server te benaderen. In principe is een file server niets anders dan een extra schijf op afstand. Meerdere gebruikers of toepassingen kunnen tegelijk gebruik maken van de bestanden die op die schijf bewaard worden.



file server-architectuur

Een gebruiker die in een LAN wil werken met data die zich op de file server bevinden, zal een DBMS moeten laden op de eigen PC. Dit locale DBMS geeft de gebruiker via de PC-interface toegang tot de data op de server. Bekende voorbeelden van dit soort producten zijn dBase en Paradox. Zodoende worden alle databasetaken op de fysieke opslag van de gegevens na, gerealiseerd op de PC. "Locking", een mechanisme om het gelijktijdig gebruik van dezelfde data te reguleren, wordt geïmplementeerd door op de file server in een bestand bij te houden wie op welk moment van welke gegevens gebruik maakt. Een "lock" (slot, afsluiting) is het ten behoeve van een bepaalde transactie ontoegankelijk maken van gegevens voor andere transacties (bij een "exclusive lock" kunnen andere transacties de gegevens niet lezen of overschrijven ; bij een "shared lock" mogen andere transacties de gegevens wel lezen, maar niet overschrijven ; in de databasetheorie wordt de omvang van een te locken eenheid de *granulariteit* of korreligheid genoemd ; de granulariteit is grof als de hele database moeten worden gelockt en zeer fijn als een afzonderlijke kolom in een rij moet worden gelockt). Zolang ieder werkstation via het lock-bestand de data benadert zijn er geen problemen. Er is echter geen mechanisme dat die werkwijze afdwingt. Omdat de controle over de database gespreid is over verschillende computers (met mogelijk verschillende besturingssystemen) is het niet eenvoudig de integriteit van de

database te waarborgen. De file server kan ook een onvolledige transactie niet terugdraaien, omdat de kennis over de transactie niet in de file server aanwezig is, maar in de PC die de transactie initieerde. Een ander probleem is de toegangsbeveiliging van de data. Wie heeft toestemming om data te lezen of te wijzigen ? Dit probleem is één van de gevolgen van het feit dat het actieve databasebeheer niet centraal bij de data plaatsvindt en dus kan omzeild worden. In een PC-netwerk zijn dit soort zaken lastig of onmogelijk op te lossen, al worden de netwerkbesturingssystemen van tegenwoordig steeds geavanceerder.

Bij de file-serverarchitectuur kan er sprake zijn van een zware belasting van het netwerk. De centrale computer (file server) bevat eigenlijk geen enkele intelligentie. Alle intelligentie is geconcentreerd in de werkstations. Door middel van de file server wordt alleen gezorgd dat bestanden door meerdere werkstations kunnen worden gemanipuleerd. Als een applicatie gegevens uit de database nodig heeft, wordt de vraagstelling op het werkstation vertaald in een aantal fysieke I/O-opdrachten naar bestanden op de file server. In het ongunstigste geval moeten complete bestanden over het LAN worden verstuurd en in het geheugen van het werkstation geladen, alvorens de selectie van de juiste data kan geschieden. Binnen deze architectuur is duidelijk geen sprake van optimaal gebruik van alle componenten ; de mogelijkheden van de PC die als file server fungeert worden nauwelijks benut.

De teleprocessing- en file server-architecturen worden veel gebruikt ten behoeve van multi-user databasesystemen, zij het voor verschillende doeleinden. Voor databases met bedrijfskritische gegevens worden nog vaak mini- of mainframe-computers gebruikt. De strikte centralisatie van data, beheer en applicaties wordt door velen nog altijd beschouwd als de enige garantie op voldoende bewaking van de gegevens. File servers worden haast uitsluitend gebruikt voor minder bedrijfskritische gegevens op afdelingsniveau. In één organisatie kunnen beide architecturen worden teruggevonden. Daarbij zijn vaak koppelingen tussen de minicomputer of het mainframe en het LAN aangebracht, waardoor PC's die opgenomen zijn in het LAN terminals kunnen emuleren. Dat betekent dat PC's zich dan kunnen gedragen als 'domme' terminals ten behoeve van toegang tot applicaties en gegevens op de minicomputer of het mainframe. Hoewel de toegang tot beide gegevensbronnen zo via dezelfde PC kan geschieden, blijven de gegevens die zich bevinden op de minicomputer en het mainframe enerzijds en de gegevens op de file server anderzijds gescheiden, op een incidentele en vaak omslachtige uitwisseling van gegevens na (import/export). Deze 'schizofrene' infrastructuur bevordert het ontstaan van een gespleten bedrijfscultuur. Aan de ene kant staat de klassieke EDP-afdeling die het beheer voert over alles dat met de minicomputer of het mainframe heeft te maken, aan de andere kant staan de gebruikers en LAN-beheerders die het recht in eigen handen hebben, omdat ze vanaf de

introductie van de PC daarvan de mogelijkheden en voordelen zijn gaan inzien.

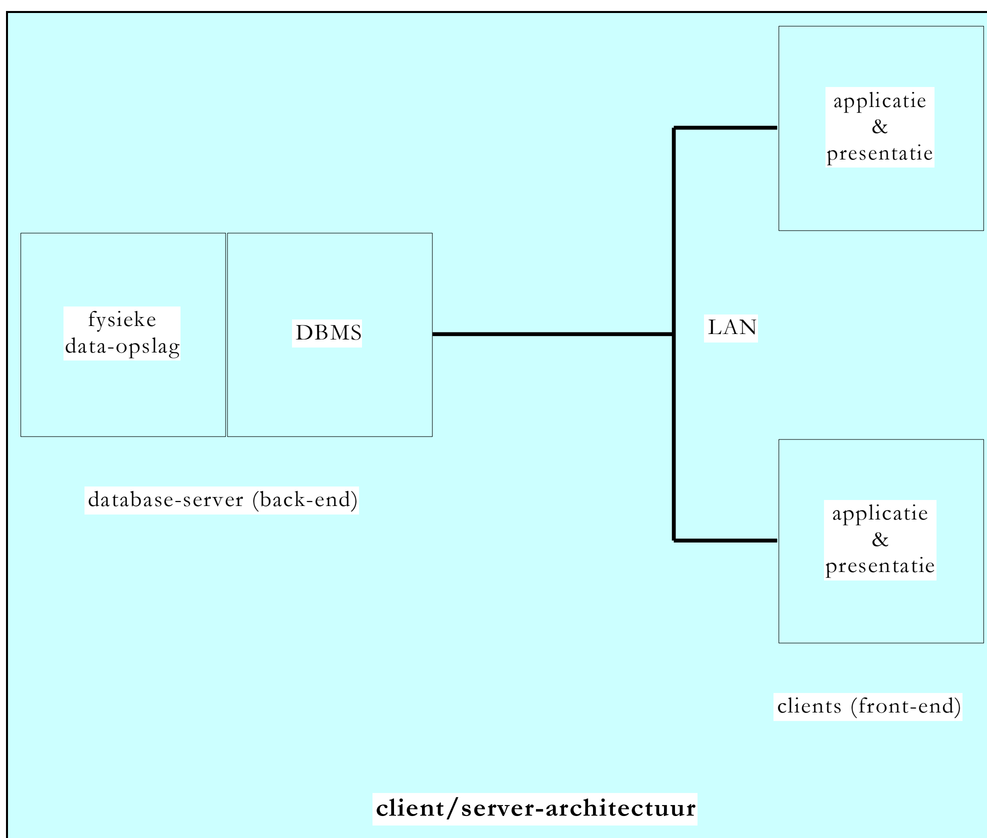
2..3. Client/server-architecturen

Het voortschrijden van de techniek biedt echter nieuwe mogelijkheden voor multi-user databasesystemen. Vooral de toename van verwerkingskracht van bureaucomputers en de ontwikkeling van meer openheid van systeemcomponenten en communicatieprotocollen, bieden perspectief. Daarbij komt dat veranderingen in bedrijfsorganisaties gewenst zijn om sneller in te kunnen spelen op marktveranderingen en effectiever te kunnen functioneren en concurreren. Verdere decentralisatie door meer zelfstandigheid en eigen verantwoordelijkheden voor afdelingen, vindt daardoor steeds meer opgang. Daarbij komt het motief bij dat een betere integratie van mini/mainframe-toepassingen en PC-LAN's gewenst is. In het ideale geval hoeft de zelfstandige PC-gebruiker zich niet of nauwelijks te realiseren dat de gegevens waarmee hij werkt misschien wel afkomstig zijn van een mainframe.

Het gevolg is een ontwikkeling van centrale computersystemen naar spreiding van verwerkingstaken en verantwoordelijkheden over meerdere (decentrale) computers. Met deze ontwikkeling naar gedistribueerde systemen deed ook *client/server* haar intrede, één van de vormen van *gedistribueerde verwerking* en een alternatief voor de hierboven besproken vormen van multi-user databasesystemen.

De client/server-architectuur bewandelt het midden tussen de mainframe- en file server-oplossing. Door naar de verdeling van de lagen van het lagenmodel te kijken, wordt duidelijk waarom. In het geval van teleprocessing-systemen zagen we dat een databasesysteem feitelijk uit maar één computer bestaat. De centrale computer is verantwoordelijk voor alle vier de taken. Deze architectuur heeft een aantal nadelen. Het centrale en hiërarchische karakter werkt bureaucratie in de hand, alsmede inefficiëntie door te weinig verantwoordelijkheid, flexibiliteit en vrijheid voor gebruikers. Er is weinig perspectief voor een betere gebruikersinterface omdat elke extra belasting van de centrale computer ten gevolge van applicaties en presentatie, direct van invloed is op de performantie van het gehele systeem. Er is bovendien een grote afhankelijkheid van de leverancier, hetgeen vaak kostenverhogend werkt en de vrijheid beperkt in het kiezen van systeemcomponenten.

Bij de file server-configuratie is er sprake van meerdere computers en gebeurt enkel de data-opslag centraal. Hieraan kleven een aantal belangrijke nadelen. De integriteit van de data kan niet worden gewaarborgd, omdat het management van de database niet centraal plaatsvindt en derhalve niet kan worden afgedwongen. Het netwerkverkeer is hoog doordat in principe elke query resulteert in het verzenden van alle voor die query relevante informatie (het DBMS bevindt zich immers op de PC en enkel het DBMS kan de relevante gegevens selecteren).



Het compromis dat client/server vormt tussen beide uitersten blijkt uit bovenstaande afbeelding. De scheiding van lagen en taken bevindt zich tussen de applicatie-logica en het DBMS. Doordat het beheer en de bewaking nu centraal bij de data op de server plaatsvindt, is er sprake van een 'intelligente' database in plaats van een 'domme' file server. Hiermee worden de voordelen van beide andere oplossingen gecombineerd, terwijl de nadelen in mindere mate of niet meer van toepassing zijn.

Net zoals bij de file server-oplossing heeft de gebruiker bij client/server de beschikking over eigen verwerkingskracht en is daarmee grotendeels onafhankelijk van de totale belasting van het computersysteem. Met de huidige stand van de techniek kan de gebruiker daardoor beschikken over een vriendelijke, grafische gebruikersinterface. Door het toevoegen van extra clients aan het netwerk, kan de (verwerkings)capaciteit vergroot worden zonder dat dit ten koste gaat van de performantie van de andere clients. Er moet wél rekening worden gehouden met de belasting per server, maar dat is geen groot probleem. Door het uitbreiden of upgraden van de server, een relatief kleine ingreep, kan de server-capaciteit vergroot worden. De client/server-architectuur biedt mogelijkheden tot de *gescheiden optimalisering van clients en de servers*, resp. *front-end* en *back-end*. Voor elk gelden andere eisen voor de combinatie van hardware, besturingssoftware en applicatie/databasesoftware. Dankzij de zelfstandigheid van clients en servers en de standaardisatie van communicatieprotocollen kan elke machine afzonderlijk worden geoptimaliseerd.

Tevens wordt het netwerkverkeer beperkt doordat over het netwerk niet langer volledige tabellen hoeven te worden verzonden. Na het ontvangen van een verzoek om data wordt door het centrale DBMS de gewenste informatie uit de onderliggende tabellen geselecteerd en wordt alleen deze informatie over het netwerk verzonden.

Doordat het DBMS zich bij de data op de server bevindt, kunnen voorwaarden of "constraints" op de data worden afgedwongen. Geen enkele applicatie heeft toegang tot de data zonder tussenkomst van het centrale DBMS. Ook is het mogelijk om transacties te coördineren en gelijktijdig gebruik van dezelfde data in goede banen te leiden. Hierdoor kunnen *de integriteit en de consistentie van de database* worden gewaarborgd.

Een ander voordeel van client/server is de mogelijkheid om dankzij standaardisatie van communicatieprotocollen heterogene configuraties te realiseren. Zo kunnen Windows-, Macintosh- en Unix-gebruikers eenvoudig is staat worden gesteld om gebruik te maken van dezelfde data, die weer afkomstig kunnen zijn van verschillende bronnen, zoals een mainframe en een database server. *Standaardisatie van communicatieprotocollen* maakt het mogelijk om de herkomst van data zo goed als onzichtbaar te maken. De gebruiker denkt en werkt in één enkele omgeving die wordt bepaald door het besturingssysteem van de client.

Een factor die op zich de complexiteit niet verhoogt, maar die toch bijdraagt tot de complexiteit, is de verantwoordelijkheid voor het succes van de overgang naar een client/server-oplossing. De kans op complicaties bij leverancier-heterogene ("multi-vendor") oplossingen is zonder meer groter dan wanneer alle systeemcomponenten van één leverancier worden betrokken. Een voorwaarde voor client/server waar dit nadeel toe leidt, laat

zich als volgt formuleren : een organisatie die client/server overweegt, zal zich er vooraf van moeten vergewissen dat voldoende kennis van *LAN-technologie, datacommunicatie en RDBMS* (relationele databasebeheer) in huis is. Bij voorkeur zal een organisatie voorafgaand aan een mogelijke strategische keuze voor client/server, in een realistische praktijksituatie experimenteren met client/server om inzicht en ervaring op te doen ten behoeve van een weloverwogen beslissing.

In hetgeen voorafging werd client/server ten tonele gevoerd als een alternatieve architectuur bij de inzet van informatietechnologie in een organisatie. Als motief voor client/server werd gewezen op technische en praktische voordelen. Deze technologie moet echter in verband worden gebracht met marktontwikkelingen en consequenties daarvan voor een organisatie. Dan blijkt client/server een "enabling technology" te zijn voor het doorvoeren van de benodigde veranderingen.

Door het vervagen van de landsgrenzen en het ten gevolge van moderne telecommunicatie- en transportmiddelen 'kleiner' worden van geografische afstanden, is een vergaande *internationalisering van de handel en dienstverlening* waar te nemen. Dit leidt tot een toename in competitie tussen leveranciers en de overgang van een "sellers market" naar een "buyers market" : de markt wordt steeds meer bepaald door de wensen en eisen van de consument. Bovendien volgen veranderingen in marktbehoeften elkaar in steeds hoger tempo op. Waar vroeger in de automobielindustrie de ontwikkeling van een nieuw model gemakkelijk tien jaar of langer kon beslaan, mag dit tegenwoordig hooguit een paar jaar zijn. Daarna loopt het ontwerp gevaar om bij oplevering al weer uit de tijd te zijn.

Om te anticiperen op deze ontwikkelingen zullen veel bedrijven hun bedrijfsvoering moeten aanpassen. In de eerste plaats dient er sneller te worden ingespeeld op veranderingen in consumentenbehoeften. Om niet achter te raken op de consument of de concurrent, is dat van nog groter belang dan vroeger. Dit tijdsaspect, wel aangeduid met "time-to-market", is de afgelopen jaren in toenemende mate centraal komen te staan. Sneller kunnen inspelen op veranderingen vereist vooral flexibiliteit bij de ontwikkeling van nieuwe diensten of producten. Ten einde deze flexibiliteit en daarmee de slagvaardigheid van een organisatie te verhogen, dient zij 'platter' en minder hiërarchisch te worden. Dit draagt immers bij aan vermindering en verkorting van communicatielijnen en beperking van "overhead" (tijd, kosten). Decentralisatie door middel van "business units" die over een grote mate van zelfstandigheid beschikken, levert meer flexibiliteit op dan afdelingen in een hiërarchie die elke verantwoordelijkheid 'naar boven' afdragen. Tevens dient bij voorkeur op projectbasis geopereerd te worden. Kleine, hooggekwalificeerde, multidisciplinaire projectteams beperken de overhead ten gevolge van communicatie en besluitvorming. Daarnaast kan het op projectbasis werken de beheersbaarheid vergroten

doordat budgetten en "deadlines" op natuurlijke wijze inzicht geven in haalbaarheid en rendement.

Om multidisciplinaire projectteams klein te kunnen houden zonder verlies van kennis, is *verbreding van kennis* noodzakelijk. Dit staat in contrast tot verdieping van kennis ofwel specialisatie. Door middel van passende opleiding kan de gewenste kennisverbreding worden bewerkstelligd, terwijl werkelijk specialistische kennis in stafafdelingen kan worden ondergebracht.

Een andere consequentie van de gesignaleerde ontwikkelingen is dat er meer *klant- en kwaliteitgericht* gewerkt zal moeten worden. Dit kan worden bereikt door 'dichter bij' de klant te opereren. Een betere terugkoppeling van de wensen en eisen van de klant bij de totstandkoming van de verleende dienst of het geleverde product is noodzakelijk. Dit komt neer op een meer continue of iteratieve ontwikkeling van diensten en producten.

De geschetste ontwikkelingen en daardoor vereiste organisatieveranderingen zijn natuurlijk ook van toepassing op de automatisering. Door de dynamiek in de ontwikkeling van informatietechnologie geldt voor deze bedrijfstak toch al dat het van groot belang is om zich van tijd tot tijd te bezinnen over mogelijkheden om efficiënter te kunnen functioneren. Maar tegen het licht van de recente marktontwikkelingen en het onderhand volwassen zijn van client/server-technologie, kunnen vele organisaties er hun voordeel mee doen om niet langer te wachten en zich nu te oriënteren op client/server[3].

HET RELATIONELE MODEL EN NORMALISATIE

1. Basisbegrippen

In de huidige maatschappij speelt kennis een grote rol. Onder *kennis* wordt verstaan datgene wat men weet over een bepaald onderwerp of gebied. Kennis kan men verwerven door fenomenen uit dat gebied waar te nemen, over het waargenomene na te denken, maar ook door uit kennisbronnen te putten. In het laatste geval maakt men gebruik van door anderen vergaarde kennis waardoor er *kennisoverdracht* plaatsvindt. Kennisoverdracht eist het expliciet maken van kennis. Kennis die overdraagbaar is noemen we *informatie* en het ontvangen van informatie zorgt in het algemeen voor *kennisvergroting*. Dit geldt zowel voor personen individueel als voor groepen van personen zoals bedrijven, ziekenhuizen, universiteiten, e.a. In een bedrijf bijvoorbeeld werken veel mensen samen en deze personen hebben gemeenschappelijke doelen. Om die doelen te bereiken, is informatie nodig.

Het overdragen van informatie (kennis) gebeurt door middel van *communicatie*. Daartoe moet de informatie gerepresenteerd worden. De gerepresenteerde informatie noemen we gegevens. De representatie kan geschieden op een vluchtige en op een permanente manier. De eerste manier houdt in dat na de communicatie de representatie verdwenen is. De *permanente representatie* (of beter : semi-permanente representatie, want niets is eeuwig), maakt het mogelijk dat de gerepresenteerde informatie meermalen gebruikt en gecommuniceerd wordt.

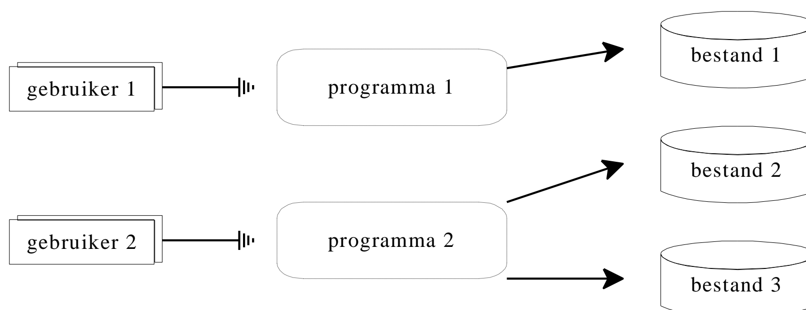
Aangezien het ondoenlijk is alle relevante informatie te onthouden, kan permanent gerepresenteerde informatie een belangrijke rol spelen bij het terugwinnen van eenmaal vergaarde, doch inmiddels vergeten kennis. De verzameling van permanent gerepresenteerde informatie noemen we de *gegevensverzameling*. Op de gegevensverzameling moet een zogenaamde 'leesfunctie' gedefinieerd kunnen worden waarmee gewenste gegevens uit de verzameling opgevraagd kunnen worden. Natuurlijk moeten ook nieuwe gegevens aan de gegevensverzameling kunnen worden toegevoegd en moeten gegevens kunnen veranderd of verwijderd worden.

Aan de bewerkingen op de gegevensverzameling kunnen vele eisen worden gesteld. Een belangrijke eis is dat de bewerkingen snel moeten kunnen worden uitgevoerd. Om aan die eis te voldoen, worden vaak computers

ingeschakeld en geschiedt de representatie van gegevens in computergeheugens.
Veronderstel dat we beschikken over een bestand met gegevens over werknemers. Het werknemersbestand kan bijvoorbeeld gebruikt worden voor het versturen van circulaires. Ook kan het bestand gebruikt worden om lijsten te produceren waaruit de leeftijdsopbouw van de werknemers blijkt. Op een bestand kunnen dus verschillende *programma's* werken. Sommige programma's werken op één bestand, andere programma's werken op meerdere bestanden. De gebruikers zullen hun wensen moeten formuleren ten aanzien van de snelheid waarmee de diverse programma's uitgevoerd moeten worden (bijvoorbeeld het adressenprogramma, het leeftijdsopbouwprogramma).

Beschouwen we nu het probleem van de opslag van het werknemersbestand. Er zijn in de loop der jaren een aantal bestandsorganisaties ontwikkeld die het mogelijk maken om records van hetzelfde type op geschikte wijze op te slaan en te manipuleren. Iedere bestandsorganisatie heeft zijn eigen karakteristieken m.b.t. de snelheid waarmee leesoperaties uitgevoerd worden ("geef de gegevens van werknemer 123456"), de snelheid waarmee bijwerkoperaties worden uitgevoerd (invoegen, verwijderen, wijzigen) en het extern geheugengebruik (sommige bestandsorganisaties hebben extra tabellen nodig, andere niet). De keuze van de optimale *bestandsorganisatie* is een moeilijk probleem.

1.1. Bestandsverwerkingssystemen



De eerste bedrijfsinformatiesystemen sloegen groepen records op in afzonderlijke bestanden en werden daarom *bestandsverwerkingssystemen* ("file-processing systems") genoemd. De bestanden die gebruikt worden in “file-processing systems” worden ook *toepassingsgebonden bestanden* genoemd,

omdat de bestanden vaak ontworpen worden voor één of meerdere toepassingen. De gegevens zijn ondergebracht in bestanden die 'eigendom' zijn van de toepassing die ze gebruikt en beheert. Gegevens worden geïsoleerd opgeslagen. Dit verschijnsel wordt *gegevensredundantie* genoemd. Hoewel bestandsverwerkingssystemen een stuk gemakkelijker zijn dan handmatige systemen, hebben ze dus hun beperkingen. Een nadeel van gegevensredundantie is dat veel meer geheugenruimte gebruikt wordt dan strikt nodig is. Een ander gevaar is dat van *inconsistentie*. Een gegevensverzameling is consistent (vrij van tegenspraak) als er niet twee (of meer) feiten uit afleidbaar zijn die met elkaar in strijd zijn. Bij gegevensredundantie wordt eenzelfde kenmerk van een entiteit meermalen opgeslagen. Hierdoor ontstaat het gevaar dat voor dat kenmerk verschillende waarden worden opgeslagen. Redundantie kan derhalve inconsistentie in de hand werken.

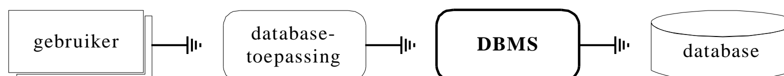
Een ander belangrijk nadeel van bestandsverwerkingssystemen is dat de programma's die op een bestand werken, beïnvloed zijn door *de keuze van de bestandsorganisatie*. De manier waarop gegevens op secundair geheugen (op schijf) zijn georganiseerd en de manier waarop toegang tot de gegevens wordt verkregen is van invloed op de programma's die op het bestand moeten werken. Kennis van de bestandsorganisatie en van de toegangstechniek moet in de programmalogica en -code verwerkt worden. Aangezien de afhankelijkheid zich afspeelt op het opslagniveau, wordt hiervoor de term *fysieke gegevensafhankelijkheid* gebruikt. Aan fysieke gegevensafhankelijkheid zijn meerdere nadelen verbonden.

Een karakter kan in EBCDIC, ASCII of een andere code gerepresenteerd zijn. Aangezien deze representatie in de programma's is opgenomen, veroorzaken veranderingen in de representatie ook veranderingen in de programmatuur. Ook indien het recordtype wijzigt, zijn aanpassingen aan de programma's nodig. Behalve programmatorische aanpassingen die een gevolg zijn van veranderingen op veld- of recordniveau, veroorzaken veranderingen op bestandsniveau eveneens programmatorische aanpassingen. Indien een bestand op één of meerdere velden geordend is, en de programma's hier ook van uitgaan, zullen deze moeten aangepast worden van zodra de ordening verandert of verdwijnt.

Gesteld kan worden dat fysieke gegevensafhankelijkheid *ongewenst* is. Waarom ? Sinds enige tijd is er een tendens dat computerapparatuur goedkoper en (automatiserings)personeel duurder wordt. Het ligt daarom voor de hand om te proberen te besparen op personeel, ook al zou dit eventueel (extra) kosten voor apparatuur met zich meebrengen. Als, zoals hierboven geïllustreerd, programma's niet onafhankelijk zijn van de opslag van de gegevens, dan is programma-onderhoud een taak die een groot beslag legt op personeel. Dat betekent dat naar een middel moet gezocht worden om af te komen van fysieke gegevensafhankelijkheid. Dit wordt nu

verkregen door een extra programmatuurlaag aan te brengen, waardoor de gegevensafhankelijkheden worden weggewerkt. Dit is de taak van databaseverwerkingssystemen.

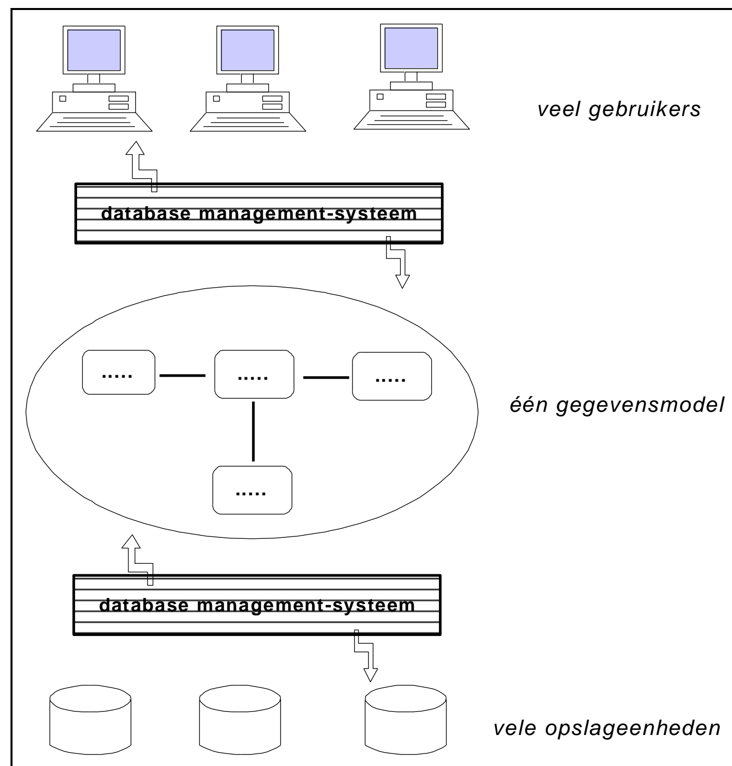
1.2. Databaseverwerkingssystemen



Een databaseverwerkingssysteem is in essentie niets anders dan een systeem dat met behulp van een computer gegevens beheert en op aanvraag ter beschikking stelt. In tegenstelling tot bestandsverwerkingssystemen, worden de gegevens niet benaderd door de toepassingen (applicaties, programma's) maar door het *DBMS* (databasemanagement-systeem). Alle aanvragen van gebruikers voor de database worden door het DBMS afgehandeld. Het DBMS moet m.a.w. *fysieke gegevensonafhankelijkheid* aanbieden. Daardoor hebben wijzigingen in de opslagstructuur geen invloed meer op programma's. Gegevensonafhankelijkheid kan worden gedefinieerd als het immuun zijn van programma's voor veranderingen in de opslagstructuur en toegangstechniek. Een eerste consequentie van fysieke gegevensonafhankelijkheid is dat de opslagstructuur en de toegangstechnieken probleemloos kunnen gewijzigd worden. Dit betekent dat de opslagstructuur en/of de toegangstechnieken o.m. als een antwoord op zich wijzigende omstandigheden kunnen veranderd worden zonder dat de bestaande toepassingen moeten aangepast worden. Deze zich wijzigende omstandigheden kunnen zijn : de komst van nieuwe (snellere) apparatuur, het belangrijk worden van voorheen als onbelangrijk beschouwde toepassingen, nieuwe gegevens die moeten worden opgenomen in de database, e.d. Deze wijzigingen hebben geen gevolgen voor de programmatuur. Een andere consequentie van fysieke gegevensonafhankelijkheid is dat toegestaan wordt dat toepassingen eenzelfde gegeven als verschillend opgeslagen beschouwen (bijvoorbeeld karakter versus numeriek).

De gegevens in de database zijn *geïntegreerd* en *gedeeld*. Deze twee kenmerken vormen het onderscheid tussen een database en een verzameling toepassingsgebonden bestanden. Geïntegreerd betekent dat de database kan worden beschouwd als de vereniging van verschillende bestanden, waarbij iedere redundantie tussen die bestanden geheel of gedeeltelijk is geëlimineerd. Met gedeeld wordt bedoeld dat delen van de database door

meerdere gebruikers mogen worden gebruikt, in die zin dat verschillende gebruikers verschillende delen van de database (kunnen) gebruiken.
Samenvattend kan gesteld worden dat het de taak is van een DBMS een brug te slaan tussen drie dingen : de gebruikers die met de gegevens willen werken, de gegevens zelf in hun correcte samenhang en de computergeheugens waarop die gegevens te vinden zijn. Onderstaand schema geeft een beeld van de architectuur van een database. Het *gebruikersniveau* omvat bijvoorbeeld hulpmiddelen die de toegang van gebruikers tot de gegevens regelen. Het middelste niveau, het *conceptueel niveau* geheten, is de “data dictionary”, die onder meer het gegevensmodel bevat. Het derde niveau wordt het *fysiek niveau* genoemd. Het omvat hulpmiddelen om de gegevens uit het model op te slaan in één of meer computergeheugens en om te zorgen dat ze snel te benaderen blijven.



De gegevens in een database worden beheerd door het DBMS. Met behulp van het DBMS kunnen databasegegevens verwerkt worden. Zonder DBMS is het onmogelijk gegevens in de database te bekijken, te wijzigen of verouderde gegevens te verwijderen. Een DBMS zal nooit uit zichzelf de gegevens in een database wijzigen of verwijderen. Met behulp van speciale talen worden opdrachten aan een DBMS gegeven. Dit soort talen wordt databasetalen genoemd. Opdrachten, ook instructies genoemd, geformuleerd

volgens de regels van de databasetaal, worden door gebruikers ingevoerd en door het DBMS verwerkt. Elk DBMS, van welke fabrikant dan ook, bezit een databasetaal. Tussen al die talen bestaan verschillen. Deze verschillende talen zijn te verdelen in groepen. Eén van deze groepen wordt gevormd door de relationele databasetalen.

SQL of "structured query language" is een relationele databasetaal. De taal bevat o.a. instructies voor het invoeren, wijzigen, verwijderen, raadplegen en beveiligen van gegevens. Deze kracht heeft geleid tot een enorme populariteit. Bestonden ze enkele jaren terug tien tot twintig SQL-producten, nu is dit aantal tot ver over de honderd uitgegroeid. SQL-producten zijn momenteel verkrijgbaar voor elk soort computer en voor elk soort besturingssysteem.

Over SQL als databasetaal zijn een paar dingen te zeggen. Omdat SQL een relationele databasetaal is, wordt zij geschaard onder de niet-procedurele databasetalen. Met 'niet-procedureel' wordt bedoeld dat gebruikers met behulp van instructies slechts hoeven op te geven met welke gegevens zij willen werken. Zij hoeven niet aan te geven hoe die gegevens één voor één benaderd moeten worden. Talen zoals C, Cobol, Pascal en Basic zijn voorbeelden van procedurele talen.

SQL is op twee manieren te gebruiken. Ten eerste kan het interactief gebruikt worden : een SQL-instructie wordt ingevoerd en direct verwerkt. Het resultaat is onmiddellijk zichtbaar. Interactief SQL is o.m. bedoeld voor eindgebruikers die databases willen benaderen. De tweede gebruikswijze is "embedded SQL". SQL-instructies zijn dan opgenomen ("embedded") in een programma dat in een andere programmeertaal geschreven is. Resultaten van SQL-instructies zijn in dit geval niet direct zichtbaar voor de gebruiker, maar worden door het omhullende programma bewerkt.

Naast het bieden van *gegevensonafhankelijkheid*, is *het handhaven van de integriteit van databasegegevens* een belangrijke taak van het DBMS. Hiermee wordt ten eerste bedoeld dat databasegegevens altijd voldoen aan regels die in de werkelijkheid gelden. Als bijvoorbeeld een werknemer maar voor één afdeling tegelijkertijd mag werken, mag nergens in de database geregistreerd worden dat een werknemer voor twee of meer afdelingen werkt. Ten tweede wordt met integriteit bedoeld dat twee verschillende databasegegevens elkaar niet tegenspreken (dit wordt ook wel consistentie van de gegevens genoemd). Elk DBMS kent instructies waarmee integriteitsregels gespecifieerd kunnen worden. Integriteitsregels behoren door een DBMS gecontroleerd te worden. Elke keer dat de inhoud van een tabel gemuteerd is, behoort het DBMS te controleren of de nieuwe gegevens aan de geldende integriteitsregels voldoen. De integriteitsregels moeten dan wel eerst gespecifieerd worden. Integriteitsregels kunnen allerlei vormen hebben. Een

aantal komt in de praktijk zo vaak voor dat ze speciale namen hebben gekregen : primaire sleutel, kandidaat-sleutel, alternatieve sleutel en refererende sleutel.

De *primaire sleutel* van een tabel is een kolom uit die tabel (of een combinatie van een aantal kolommen) die gebruikt kan worden als unieke identificatie voor de rijen in die tabel. Met andere woorden : twee verschillende rijen in een tabel mogen nooit in de primaire sleutel dezelfde waarde hebben en in elke rij moet de primaire sleutel altijd één waarde hebben. Sommige tabellen bezitten meer dan één kolom (of combinatie van kolommen) die aangewezen zou kunnen worden als primaire sleutel. Zij bezitten allemaal de eigenschappen van een primaire sleutel. Deze kolommen (of combinaties van kolommen) worden *kandidaat-sleutels* genoemd. Slechts één daarvan wordt als primaire sleutel aangewezen. Een kandidaat-sleutel die niet de primaire sleutel van een tabel vormt, wordt een *alternatieve sleutel* genoemd. Een *refererende sleutel* is een kolom (of combinatie van kolommen) in een tabel die een deelverzameling is van de primaire sleutel van een tabel. Andere namen voor refererende sleutel zijn : verwijzende sleutel, vreemde sleutel of "foreign key".

2. Definitie van het begrip database

Het begrip 'database' kunnen we op twee manieren beschouwen. Ten eerste kunnen we een database zien als een verzameling gegevens die fysiek op een disk is opgeslagen. Dit is een technische beschouwing van het begrip. Ten tweede kunnen we databases zien als 'logische' gegegevensverzamelingen ongeacht de wijze waarop de gegevens geregistreerd worden. In dit opzicht kan een database omschreven worden als *een zichzelf beschrijvend geIntegreerd geheel van gegevensverzamelingen, bestemd voor informatieverstrekking in ruime kring en beheerd door een databasemanagement-systeem.*

Een database beschrijft zichzelf. Naast gegevens bevat een database ook een beschrijving van zijn eigen structuur. Deze beschrijving wordt een "data dictionary" (of : metadata) genoemd. De "data dictionary" maakt de eigenschap van gegevensonafhankelijkheid mogelijk. In dit opzicht lijkt een database op een bibliotheek. Een bibliotheek is een zichzelf beschrijvende verzameling boeken. Behalve de boeken zelf bevat de bibliotheek een catalogus waarin de boeken beschreven worden.

Waarom is het zo belangrijk dat een database een beschrijving van zichzelf bevat ? Ten eerste wordt de gegevensonafhankelijkheid erdoor vergroot. De structuur en de inhoud van de database kan uit de database zelf worden

gehaald. Externe informatie over bestands- en recordopmaak wordt niet bijgehouden. Ten tweede : als de structuur van de gegevens in de database verandert (bv. toevoegen van een nieuw veld), dan moet die wijziging enkel opgenomen worden in de "data dictionary". Verder bevat een database indexen en gegevens over de applicaties die de database gebruiken. Deze laatste categorie gegevens noemen we *applicatie-metadata*.

3. Database-terminologie

De terminologie die gebruikt wordt voor de formele beschrijving van een database en de onderdelen waaruit die is opgebouwd, is afkomstig uit verschillende bronnen. De uitdrukkingen voor de beschrijving van gegevens, zoals gegevenseenheid en attribuut, zijn afkomstig uit de terminologie van de statistiek. Een andere reeks termen, die voor de beschrijving van dezelfde onderdelen worden gebruikt, is gebaseerd op de computerterminologie en heeft betrekking op de manier waarop deze onderdelen op schijf opgeslagen worden (gegevensopslag). De querymethode (QBE : *Query By Example*) en ook SQL (*Structured Query Language*) gebruiken eveneens de terminologie voor gegevensopslag. De terminologie van de statistiek en de terminologie voor gegevensopslag kunnen vergeleken worden met de terminologie die gebruikt wordt bij de objectgeoriënteerde programmeermethode (OOP : *Object Oriented Programming*).

Een database bevat gegevens over *objecten* die werkelijk bestaan. Dat zijn bijvoorbeeld mensen, boeken uit een bibliotheek, facturen of bestelformulieren, e.d. Dergelijke objecten zijn tastbaar. Elk object heeft een fysieke verschijning, al is het maar het beeld op een computerscherm. Tastbare objecten hebben eigenschappen en gedragingen. Deze combinatie lijkt misschien alleen van toepassing op databases van personen, niet van boeken. Alle database-objecten, behalve die in archiefdatabases, hebben echter zowel eigenschappen als gedragingen. Archiefdatabases worden gebruikt voor de opslag van gegevens die nooit veranderen ; aan dergelijke databases worden enkel nieuwe gegevens toegevoegd.

De eigenschappen van een object bepalen de inhoud van een database of een tabel die bepaalde voorstellingen van objecten van hetzelfde type bevat. Zo wordt aan boeken een ISBN-nummer toegekend. Dit nummer is een eigenschap van het object 'boek', net zoals de titel, de auteur, het aantal pagina's, het soort omslag ,e.a. Dergelijke eigenschappen zijn statisch : ze blijven dezelfde. Het object 'bankrekening' heeft een aantal statische eigenschappen zoals rekeningnummer en rekeninghouder. Het saldo van

een bankrekening is een dynamische eigenschap van het object 'bankrekening'.

Het werkelijk bestaande object is in eerste instantie de bron van informatie die in de database als een gegevenseenheid wordt weergegeven. De feitelijke representatie van een object in een database wordt een *gegevenseenheid* genoemd. Een afzonderlijke gegevenseenheid moet uniek zijn, zodat deze van andere gegevenseenheden kan onderscheiden worden. Een bankrekening is een gegevenseenheid met een unieke aanduiding (het bankrekeningnummer). Een gegevenseenheid is m.a.w. een unieke representatie van één reëel object, gecreëerd met behulp van de waarden van de eigenschappen van het object in een vorm die voor de computer leesbaar is. Een gegevenseenheid komt overeen met een rij in QBE en SQL, of een record in de terminologie van gegevensopslag. Gegevenseenheden worden ook *gegevensentiteiten*, *gegevensobjecten* of *gegevensinstanties* genoemd.

Een *attribuut* is een significante eigenschap van een object. Attributen bevinden zich in velden (terminologie van gegevensopslag) of kolommen (QBE en SQL). Een attribuut wordt ook wel een cel of gegevenscel genoemd. Met deze termen wordt een snijpunt tussen een rij en een kolom of het snijpunt van een veld en een record aangeduid.

Het *gegevenstype van attributen* verwijst naar numerieke (integer, drijvende komma) of alfanumerieke gegevenstypen (letters, cijfers, speciale tekens). De toegestane reeks waarden voor een attribuut wordt het *attribuutdomein* genoemd. *Attribuutwaarden* worden beperkt tot de waarden die binnen het domein van het attribuut liggen (een attribuutwaarde is de kleinste ondeelbare eenheid van gegevens binnen een gegevenseenheid ; de termen celwaarde en gegevenswaarde zijn synoniemen voor attribuutwaarde). Er wordt gebruik gemaakt van een aantal regels om de integriteit van de gegevens te waarborgen. Met behulp van *validatieregels* wordt bepaald welke waarden voor een veld geldig zijn of binnen welk bereik de waarde moet vallen. Geavanceerde systemen zijn in staat relaties te definiëren tussen verschillende verzamelingen gegevens (tabellen) ; het DBMS moet ervoor zorgen dat geen gegevens worden ingevoerd, veranderd of gewijzigd die strijdig zijn met de relaties in de database (in Microsoft Access wordt dit mogelijk gemaakt door het afdwingen van *referentiële integriteit*).

Om een gegevenseenheid uniek te identificeren, wordt gebruik gemaakt van één of meerdere attributen (velden). Er wordt gesproken van *primaire sleutels*.

De verzameling van alle gegevenseenheden van eenzelfde type wordt een *homogeen universum* genoemd. De gegevenseenheden moeten beschikken over een identieke verzameling van attributen. Deze verzameling komt overeen met een *tabel* van een database en wordt ook een *eenheidsklasse* of *eenheidstype* genoemd. De leden van een homogene verzameling worden

soms als *eenheidsinstanties* of gewoon *instanties* aangeduid. De verzameling van aanverwante eenheidsklassen (= aanverwante homogene universums) wordt een *heterogeen universum* genoemd. Een heterogeen universum komt overeen met een *database*, bestaande uit aanverwante tabellen.

Vrijwel alle moderne database managementsystemen bewaren en verwerken informatie in een *relationeel gegevensmodel*. De term 'relationeel' duidt op het feit dat ieder record (rij) in de database informatie bevat die betrekking heeft op een bepaald onderwerp (object) en alleen op dat onderwerp. In een relationeel databasemanagement-systeem, soms ook kortweg *RDBMS* genoemd, worden gegevens opgeslagen in tabellen. Tabellen bevatten informatie over een bepaald onderwerp en bestaan uit kolommen (velden) waarin verschillende gegevens over het betreffende onderwerp zijn ondergebracht (attributen, attribuutwaarden, celwaarden).

Het relationele model is een bepaalde manier van denken over en omgaan met gegevens. Het kent drie aspecten : structuur, integriteit en manipulatie. '*Structuur*' geeft aan welke entiteiten, attributen en associaties uit de waarneembare wereld ("universe of discourse") in een database geregistreerd worden. In het relationele model worden entiteiten, attributen en associaties met zgn. *relaties* weergegeven. Een relatie bevat gegevens over entiteiten, bijvoorbeeld over werknemers, met gegevens over attributen zoals naam, adres, leeftijd e.d. Een relatie is enigszins te vergelijken met een bestand. Maar een bestand mag niet verward worden met *associaties tussen entiteiten*, die eveneens door een relatie kunnen weergegeven worden. Zo kan de relatie 'werknemer' gegevens over de attributen van de eniteit 'werknemer' bevatten en kan de relatie 'project' gegevens bevatten over de associatie tussen de entiteit 'werknemer' en de projecten waaraan werknemers werken.

'*Integriteit*' geeft aan, aan welke eisen de gegevens moeten voldoen. Gegevens in een relatie moeten altijd aan een aantal regels voldoen. Een werknemer kan bijvoorbeeld maar voor één afdeling werken of het salaris van een werknemer mag niet hoger zijn dan 150.000 fr. Dit soort regels worden integriteitsregels genoemd. Het doel van *integriteitsregels* is ervoor te zorgen dat gegevens elkaar nooit tegenspreken en te allen tijde in overeenstemming met de werkelijkheid zijn. Er mag niet vergeten worden dat gegevens in een relatie een weergave of een model van de werkelijkheid zijn.

'*Manipulatie*' geeft aan hoe de gegevens verwerkt kunnen worden (toevoegen, verwijderen, wijzigen en selecteren van gegevens). In dit opzicht kent het relationele model *expressies* voor het manipuleren van gegevens. Hierbij wordt een onderscheid gemaakt tussen expressies voor het selecteren van gegevens uit relaties en expressies voor het wijzigen van gegevens in relaties.

De functies van relationele expressies kunnen vergeleken worden met zoek- en mutatie-opdrachten in een databasetaal (zoals SQL).

Het is belangrijk voor ogen te houden dat het relationele model geen regels bevat over *hoe* gegevens fysiek opgeslagen moeten worden (in welke volgorde en op welke locatie). Het relationele model beschrijft ook niet *hoe* expressies moeten verwerkt worden om tot een resultaat te komen, of *hoe* integriteitsregels geïmplementeerd of gecontroleerd moeten worden. Al deze taken behoren toe aan het databasemanagement-systeem.

4. Ontwerp en opzet van databases

Een database is een model van het gebruikersmodel van de werkelijkheid. Om een geschikte databasetoepassing te kunnen maken, moeten de ontwerpers en ontwikkelaars eerst een volledig beeld hebben van dit gebruikersmodel. De eerste stap bij het ontwerpen van een database is het bepalen welke objecten in de database worden weergegeven en welke eigenschappen van die objecten worden opgenomen. Dit proces noemen we een *gegevensmodel* opstellen. Het doel van een gegevensmodel is het creëren van een logische representatie van de gegevensstructuur die wordt gebruikt om een database te maken.

Een model waarin de objecten weergegeven worden, noemen we een *conceptueel gegevensmodel*. Dit in tegenstelling tot de feitelijke tabellen die later van de objecten gemaakt worden.

Databases kunnen volgens verschillende strategieën ontworpen worden. De twee belangrijkste stromingen op het gebied van het ontwerpen van databases zijn het *procesgericht ontwerpen* (ook wel *top down-methode* genoemd) waarbij men zich richt op de functies en taken die moeten uitgevoerd worden en het *gegevensgericht ontwerpen* (ook *bottom up-methode* genoemd) waarbij men zich concentreert op het achterhalen en ordenen van de gegevenselementen die nodig zijn. Deze twee methoden leveren databases met een totaal verschillende structuur op. De bottom up-methode levert *toepassingsdatabases* op en de top down-methode levert *onderwerpdatabases* op.

4.1. Toepassingsdatabases

Gegevensmodellen kunnen gebaseerd worden op specifieke behoeften om gegevens op een bepaalde manier weer te geven. Voor een dergelijk op behoeften gebaseerd model wordt o.m. vertrokken van de weergave van de gegevens op een scherm, in een afgedrukt rapport, e.d.

Indien men een eenvoudige database voor eigen gebruik of één soort gegevensobject ontwerpt, is de bottom up-methode geschikt. Het probleem van de bottom up-methode is dat de mogelijkheid bestaat dat er meerdere afzonderlijke databases met dubbele gegevens worden gecreëerd. Verschillende personen of groepen binnen een organisatie hebben misschien een toepassingsdatabase nodig die bijvoorbeeld een klantenlijst bevat. Als in één toepassingsdatabase een nieuwe klant wordt toegevoegd of de gegevens van een bestaande klant worden gewijzigd, moeten alle andere toepassingsdatabases eveneens worden bijgewerkt. Dit bijwerken kost veel tijd en kan fouten opleveren.

4.2. Onderwerpdatabases

Bij de top down-methode gaat men van het algemene naar het specifieke. Men begint bijvoorbeeld met het formuleren van een aantal strategische bedrijfsdoelstellingen en stelt vervolgens vast welke gegevens aan het bereiken van die doelstellingen kunnen bijdragen. Resultaat van dit onderzoek is een abstract gegevensmodel dat vervolgens meer en meer gedetailleerd en verfijnd wordt totdat duidelijk is hoe de database en de bijhorende applicaties eruit moeten zien.

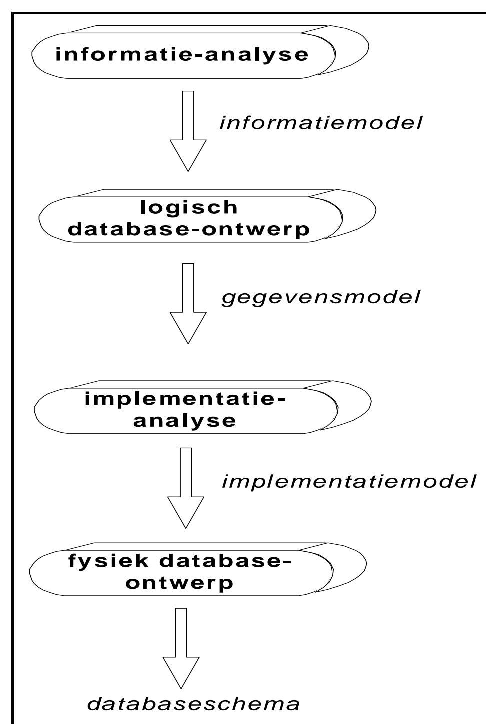
Databases die bestaan uit tabellen die betrekking hebben op onderwerpen of functies uit één klasse, worden onderwerpdatabases genoemd. In dit geval wordt het ontwerp niet bepaald door de toepassingen waarin de gegevensobjecten worden gebruikt, maar door de eigenschappen van de objecten zelf. Voor het ontwerpen van een onderwerpdatabase wordt vertrokken van het definiëren en beschrijven van alle relevante objecten en de relaties tussen die objecten.

Voorstanders van de top down-methode stellen dat deze beter is dan de bottom up-methode omdat gegevensmodellen worden uitgewerkt vanuit het gezichtspunt van de hele organisatie. Voorstanders van de bottom up-methode verkiezen deze methode boven de top down-methode omdat de bottom up-methode sneller is. Zij stellen dat de top down-methode leidt tot

omvangrijke rapporten die vervolgens in een la terechtkomen. Er kunnen wegens de grote hoeveelheden te analyseren gegevens 'verlammingsverschijnselen' binnen het project optreden, de zgn. analyse-paralyse. Hoewel de bottom up-benadering misschien niet het beste systeem voortbrengt, leidt deze aanpak in ieder geval snel tot een tastbaar resultaat. Men kan veel sneller gebruik maken van het systeem.

Of men nu een top down- of bottom up-benadering kiest, in beide gevallen dient een gegevensmodel opgesteld worden. Het opstellen van het gegevensmodel staat centraal bij het *ontwerp* van databases. Eens bepaald is welke gegevenselementen moeten verzameld worden, welke de onderwerpen zijn waarvan de gegevenselementen deel uitmaken en welke de relaties zijn tussen de verschillende onderwerpen, kan een databaseschema opgesteld worden. Een *databaseschema* is een grafische weergave van de indeling van tabellen, in de vorm van balken die de veldnamen bevatten en een vereenvoudigde weergave van de relaties tussen die velden. Vervolgens kan met het feitelijke uitvoerende werk, nl. het aanmaken van tabellen, en dus de *opzet* van de database begonnen worden.

M.b.t. het ontwerp en de opzet van databases wordt in de literatuur doorgaans een onderscheid gemaakt tussen een viertal fasen. In onderstaand schema worden deze fasen voorgesteld.



Om te komen tot een databaseschema voor een database moet eerst bekend zijn welke gegegevens geregistreerd moeten worden. M.a.w. : alle informatiebehoeften moeten geïnventariseerd worden. Het bepalen van alle relevante gegevens wordt *informatie-analyse* genoemd. Het resultaat is een *informatiemodel*. Tijdens informatie-analyse worden o.m. mensen geïnterviewd, formulieren geanalyseerd en processen bestudeerd.

Tijdens de fase van logisch database-ontwerp wordt een informatiemodel omgezet naar een gegevensmodel. In een gegevensmodel wordt aangegeven welke gegevens in de database opgeslagen moeten worden (een informatiemodel bevat ook specificaties die niet in de database terechtkomen). Er wordt nog niet bepaald hoe en waar de gegevens opgeslagen moeten worden. *Logisch database-ontwerp* wordt daarom ook *conceptueel database-ontwerp* genoemd.

Tijdens de twee laatste fasen worden aan het conceptuele gegevensmodel allerlei technische en fysieke specificaties toegevoegd om een effectieve database te kunnen creëren. Er wordt bijvoorbeeld bepaald hoe (al of niet gesorteerd, met of zonder index) en waar (op welke disk) de gegevens opgeslagen zullen worden. Het resultaat is een databaseschema [4].

4.3. Logisch database-ontwerp

Het is, gegeven een informatiemodel, eenvoudig een gegevensmodel te ontwikkelen, want het is altijd mogelijk een relatie met een groot aantal attributen te creëren waarin alle gegevens geregistreerd kunnen worden. Echter, het wordt zeer moeilijk om de waarden in de relatie te muteren. Er kunnen toevoeg-, verwijder- en/of update-anomalieën ontstaan vanwege de aanwezigheid van *redundantie in de relatie*. Bij het ontwerpen van relaties moet ernaar gestreefd worden de hoeveelheid redundantie te minimaliseren. Vandaar dat logisch database-ontwerp kan gedefinieerd worden als het ontwerpen van een gegevensmodel op basis van een informatiemodel, waarbij ernaar gestrefd wordt elk feit éénmaal te registreren om zodoende redundantie te vermijden en daardoor problemen te voorkomen bij het toevoegen, verwijderen en wijzigen van gegevens. Om redundantie in de relatie te voorkomen, staat *de analyse van afhankelijkheden tussen attributen* centraal bij logisch database-ontwerp.

Het *normaliseren van relaties* is één van de bekendste technieken voor het omzetten van een informatiemodel naar een gegevensmodel. De theorie achter normaliseren is omvangrijk en complex, maar de tests die kunnen uitgevoerd worden om te controleren of een ontwerp logisch en gemakkelijk

te gebruiken is, zijn redelijk eenvoudig en kunnen worden geformuleerd in de vorm van regels (cfr. infra).

Het relationele model is om twee redenen belangrijk. Ten eerste is het relationele model robuust en algemeen en kan het gebruikt worden voor DBMS-onafhankelijk ontwerp. Ten tweede vormt het relationele model de grondslag van een grote groep DBMS-producten. Bekendheid met het relationele model zal implementatie van databases met behulp van deze producten vergemakkelijken.

In hetgeen volgt, worden de grondslagen van het relationele model en van het proces van normalisatie behandeld. Normalisatie is een proces dat een relatie die bepaalde problemen bevat, vertaalt naar twee of meer relaties die deze problemen niet bevatten. Over de vraag wat een goed gestructureerde relaties is, is veel onderzoek verricht. De basis van dit onderzoek is het werk van *E.F.Codd*, die normaalvormen voor relaties definieerde [5].

4.3.1. Relaties

Een relatie is een *tweedimensionele tabel*. De rijen bevatten gegevens over een grootheid of een onderdeel van een grootheid. De kolommen bevatten gegevens over een bepaalde eigenschap van die grootheid. De rijen worden tupels genoemd en de kolommen attributen. De termen relatie, tupel en attribuut stammen uit de relationele algebra die het relationele model theoretisch onderbouwt. Veel mensen gebruiken liever bestand, record en veld en anderen verkiezen tabel, rij en kolom.

De term *relatie* heeft een nauwkeuriger definitie dan de term bestand of tabel. De relationele visie van gegevens is gebaseerd op de observatie dat bestanden die aan zekere restricties voldoen, beschouwd mogen worden als relaties. Een tabel is pas een relatie als aan bepaalde voorwaarden voldaan is. Ten eerste moeten de velden per rij alle *enkelwaardig* zijn. Ten tweede moeten alle waarden in eenzelfde kolom eenzelfde soort gegeven voorstellen. De kolommen hebben unieke namen en hun volgorde in de tabel heeft geen specifieke betekenis. Ook de volgorde van de rijen is zonder betekenis, maar twee rijen waarin alle waarden precies hetzelfde zijn (identieke tupels), zijn niet toegestaan.

Relaties bezitten belangrijke eigenschappen.

1. *Er zijn geen duplicaattupels*. Deze eigenschap volgt uit het feit dat de extensie van een relatie een wiskundige verzameling is (nl. een verzameling van tupels) en een verzameling kent per definitie geen duplicaten. Tupels zijn uniek. De *uniciteitseigenschap* volgt uit het feit dat de waarden van attributen een tupel uniek bepalen, d.w.z. identificeren. Normaliter is het niet nodig alle attributen erbij te betrekken ; een kleinere combinatie is gewoonlijk voldoende. Een domein is een verzameling van waarden waaruit de actuele waarden moeten komen die een attribuut mag aannemen. Domeinen zijn dus verzamelingen van waarden, waaruit de attributen hun actuele waarden putten.

2. *Tupels zijn ongeordend*. Deze eigenschap volgt eveneens uit het feit dat de extensie van een relatie een wiskundige verzameling is. Verzamelingen zijn nu eenmaal niet geordend. Het is enigszins ongelukkig dat relaties op papier in de vorm van tabellen worden gerepresenteerd, omdat deze representatie suggereert dat er een ordening is.

3. *Attributen zijn ongeordend*. Deze eigenschap volgt uit het feit dat een relatie ook als een verzameling attributen is gedefinieerd. Zoals met vorige eigenschap wordt door de tabelrepresentatie van een relatie gesuggereerd dat de attributen geordend zijn. De attributen van een relatie zijn evenwel ongeordend.

4. *Alle attribuutwaarden zijn atomair*. Deze eigenschap volgt uit het feit dat domeinen alleen atomaire waarden bevatten. Anders geformuleerd : op iedere rij-kolompositie in een tabelrepresentatie mag slechts één waarde voorkomen, nooit een verzameling van waarden. Een relatie die eigenschap 4 bezit, is in eerste normaalvorm (1NV), of anders gezegd, is genormaliseerd.

Om het relationele model en het normalisatieproces goed te begrijpen, dienen we de betekenis van de begrippen functionele afhankelijkheid en sleutel uit te leggen. *Functionele afhankelijkheid* wordt veroorzaakt door een onderling verband tussen attributen. Als we de waarde van een attribuut op de een of andere manier kunnen bepalen of opzoeken als we de waarde van een ander attribuut kennis, is er sprake van functionele afhankelijkheid. Algemener : attribuut Y wordt functioneel afhankelijk van attribuut X als door de waarde van X die van Y bepaald is (anders geformuleerd : bij elke verschillende waarde van X hoort slechts één waarde van Y). Functionele afhankelijkheid is dus een samenhang tussen attribuutwaarden.

Een *sleutel* is een groep van één of meer attributen waardoor een rij in een tabel uniek wordt geïdentificeerd. Iedere relatie heeft minstens één sleutel. Soms wordt een sleutel gevormd door één enkel attribuut en soms wordt een sleutel gevormd door een aantal attributen samen.

Om records in een tabel terug te vinden en records uit verschillende tabellen logisch met elkaar te verbinden, worden speciale velden gebruikt : *sleutelvelden*. Er zijn twee belangrijke soorten sleutels : primaire sleutels en vreemde (= verwijzende) sleutels.

Een *primaire sleutel* ("primary key") van een tabel is een kolom uit die tabel die gebruikt kan worden als unieke identificatie voor de rijen in die tabel. M.a.w. twee verschillende rijen in een tabel mogen nooit dezelfde primaire sleutelwaarde hebben. Als de gegevens uit meer dan één kolom van een tabel moeten gebruikt worden om de rijen van een tabel uniek te kunnen identificeren, spreken we van een *samengestelde primaire sleutel*. Sommige tabellen hebben meer dan één kolom die aangewezen zou kunnen worden als primaire sleutel. Deze bezitten allemaal de eigenschappen van een primaire sleutel. Slechts één enkele kolom of een bepaalde combinatie van kolommen wordt aangewezen als primaire sleutel. Een tabel heeft minimaal één kandidaat-sleutel (“candidate key”). Per relatie kiezen we een primaire sleutel. De andere (kandidaat)sleutels worden dan *alternatieve sleutels*.

Een vreemde sleutel (“foreign key”) is een kolom of een combinatie van kolommen waarvan de inhoud telkens verwijst naar de overeenkomstige waarde van een primaire sleutel in een andere tabel.

4.3.2. Normalisatie

In het relationele model voldoet iedere relatie aan de voorwaarde dat iedere attribuutwaarde atomair is. Zo'n relatie is in eerste normaalvorm (1NV). Het is mogelijk om verdere niveaus van normalisatie te definiëren : een tweede normaalvorm (2NV), derde normaalvorm (3NV), enz. Een relatie is in 2NV als deze in 1NV is en bovendien nog aan een extra voorwaarde voldoet. Elke normaalvorm is meer restrictief dan de vorige. Belangrijker is echter dat iedere normaalvorm in het algemeen wenselijker is dan de vorige. De database-ontwerper moet het verkrijgen van relaties in een hoge normaalvorm niet als wet zien, maar interpreteren als richtlijn. De enige harde eis is, dat de relaties tenminste in eerste normaalvorm gebracht moeten worden. Voor grote databases, d.w.z. met veel relaties, is database-ontwerp een zeer complexe taak waarbij normalisatietechnieken behulpzaam kunnen zijn.

Het globale doel van (verdere) normalisatie is redundantie weg te werken. De normalisatiediscipline voorziet in een verzameling richtlijnen waarmee een relatie met bepaalde redundanties kan worden afgebroken in kleinere relaties die de redundantie niet bezitten. Het uiteindelijke doel is te eindigen

met een database-ontwerp waarin ieder feit slechts op één plaats voorkomt (zodat geen redundantie aanwezig is). Normalisatie houdt zich bezig met het reduceren van grotere relaties tot kleinere, waarbij redundanties worden weggewerkt. Daarbij wordt er a.h.w. van uitgegaan dat er een (kleine) verzameling van grotere relaties voorhanden is. Deze verzameling wordt zodanig bewerkt, dat een grotere verzameling kleinere relaties overblijft.

Niet alle structuren van relatietabellen zijn even wenselijk. Een tabel die voldoet aan de minimale eisen die nodig zijn om de tabel een relatie te noemen, kan toch ongeschikt zijn. Bij bepaalde relatiestructuren hebben wijzigingen van gegevens ongewenste effecten tot gevolg. Men spreekt van modificatie-anomalieën (bv. verwijderanomalieën, invoeganomalieën). Een anomalie (onregelmatigheid) kan worden verwijderd door een relatie te splitsen in 2 of meer relaties. Dit proces wordt normalisatie genoemd.

Relaties kunnen geclassificeerd worden naar het type anomalie waarvoor ze gevoelig zijn. In de jaren zeventig is hiervan een uitgebreide studie gemaakt. Telkens als een nieuwe anomalie gevonden werd, werd deze geclassificeerd en werd een methode bedacht om deze te voorkomen. Deze relatieklassen worden normaalvormen genoemd. E.F. Codd definieerde in 1970 een eerste, tweede en derde normaalvorm (1NV, 2NV, 3NV). Later werden nog de Boyce- Codd-normaalvorm (BCNV) en een vierde en vijfde normaalvorm (4NV, 5NV) gedefinieerd.

De normaalvormen bleken een nuttig hulpmiddel, maar hebben toch een ernstige tekortkoming : niets garandeert dat gegevens in een bepaalde normaalvorm geen anomalie meer bevatten. Iedere normaalvorm elimineert slechts enkele specifieke anomalieën. In 1981 kwam er verbetering in deze situatie : R.Fagin definieerde een nieuwe normaalvorm, de domein/sleutel-normaalvorm (DS/NV)[6]. Fagin toonde aan dat een relatie in domein/sleutel-normaalvorm geen modificatie-anomalieën bevat. Ook bewees hij dat iedere relatie zonder modificatie-anomalieën in domein/sleutel-normaalvorm is. Door Fagins bewijs was het niet meer nodig telkens opnieuw te zoeken naar nieuwe anomalieën en steeds nieuwe normaalvormen te bedenken. Als een relatie in DS/NV is dan weten we dat deze geen anomalieën bevat, maar de grote vraag is nu hoe een relatie in DS/NV gezet wordt.

4.3.2.1. Normaalvormen

4.3.2.1.1. De eerste normaalvorm

Regel 1 - Unieke velden : elk veld in een tabel moet worden gebruikt voor een uniek type informatie

Iedere tabel die voldoet aan de minimale eisen die aan een relatie worden gesteld, is in eerste normaalvorm. Alle velden in de tabel moeten enkelwaardig zijn. Alle elementen in een bepaalde kolom moeten hetzelfde soort gegeven voorstellen. De kolommen moeten unieke namen hebben en er mogen geen twee rijen voorkomen met volledig identieke gegevens. De volgorde van rijen en kolommen heeft geen verdere betekenis.

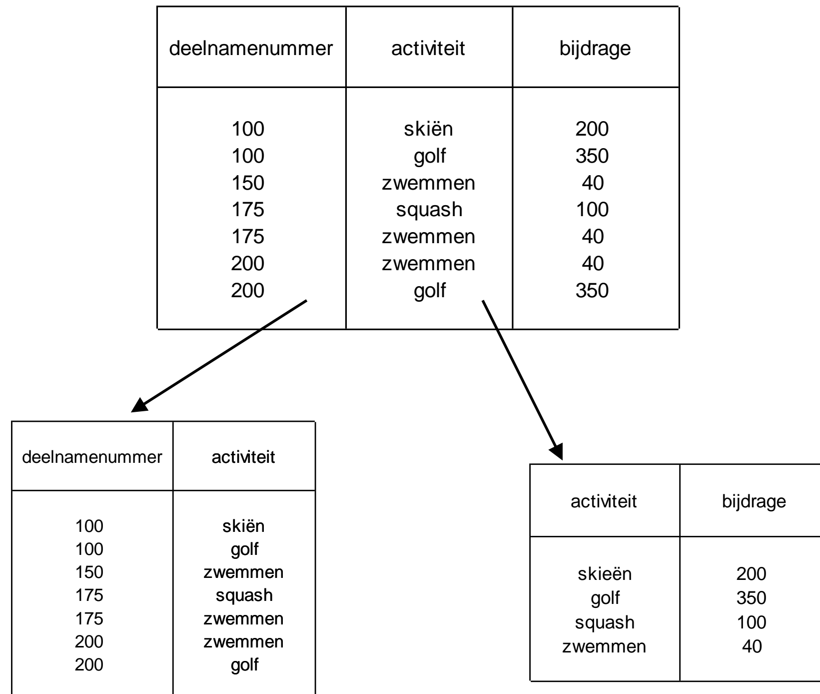
4.3.2.1.2. De tweede normaalvorm

Regel 2 - Elke relatie moet één of meerdere kolommen (velden) bevatten die records uniek identificeren (primaire sleutel). Daarbij dienen alle niet-sleutelvelden in een tabel afhankelijk te zijn van de gehele sleutel.

Nemen we als voorbeeld de relatie (zie hieronder) waarin naast het deelnamenummer ook de activiteit en de bijdrage ervan als velden opgenomen zijn. Deze relatie is onderhevig aan *modificatie-anomalieën*. Het verwijderen van de deelnemer die squash als activiteit kiest, betekent dat we de bijdrage van squash verliezen. Het is anderzijds ook niet mogelijk een activiteit toe te voegen voordat er iemand op inschrijft. Het probleem is te wijten aan het feit dat er *afhankelijkheid tussen gegevens* voorkomt waarbij maar een deel van de samengestelde sleutel betrokken is.

In onderstaande tabel is de sleutel (deelnamenummer, activiteit) en de relatie bevat de *functionele afhankelijkheid* (activiteit -> bijdrage). Bijdrage is partieel afhankelijk van de sleutel. De genoemde modificatie-anomalieën zouden niet bestaan indien bijdrage van de gehele sleutel zou afhangen. Daarom moet de relatie gesplitst worden in twee relaties.

sleutel met 2 attributen (deelnamenummer, activiteit)



deelnamenummer	activiteit	bijdrage
100	skiën	200
100	golf	350
150	zwemmen	40
175	squash	100
175	zwemmen	40
200	zwemmen	40
200	golf	350

deelnamenummer	activiteit
100	skiën
100	golf
150	zwemmen
175	squash
175	zwemmen
200	zwemmen
200	golf

activiteit	bijdrage
skieën	200
golf	350
squash	100
zwemmen	40

De tweede normaalvorm kan dus als volgt gedefinieerd worden : *een relatie heeft de tweede normaalvorm als alle attributen die niet in de sleutel zijn opgenomen van de gehele sleutel afhankelijk zijn*. Dit impliceert eveneens dat relaties met een sleutel die maar uit één attribuut bestaat, automatisch de tweede normaalvorm hebben.

4.3.2.1.3. De derde normaalvorm

Regel 3 - Een relatie mag geen transitieve afhankelijkheden bevatten.

Een relatie is in derde normaalvom als de relaties in tweede normaalvorm is en *geen transitieve afhankelijkheden* bevat.

nummer	buurt	huurprijs
100	Uilenstede	500
150	Kattenburg	480
200	Uilenstede	500
250	Krakeelhof	460
300	Uilenstede	500

nummer => buurt => huurprijs

Om anomalieën die te wijten zijn aan transitieve afhankelijkheden uit een relatie in tweede normaalvorm te verwijderen, moeten de transitieve afhankelijkheden weggewerkt worden door het opsplitsen van de relatie (een relatie met nummer en buurt als attributen en nummer als sleutel en een relatie met buurt en huurprijs als attributen en buurt als sleutel).

4.3.2.1.4. De Boyce-Codd normaalvorm

Regel 4 - In een relatie moet elke determinant ook kandidaatsleutel zijn.

Tot nu toe hebben we relaties beschouwd die slechts één kandidaatsleutel hebben. Eventueel mag deze kandidaatsleutel samengesteld zijn, d.w.z. bestaan uit een combinatie van attributen. Het blijkt echter dat problemen kunnen optreden zodra een relatie *meer dan één kandidaatsleutel* heeft, die kandidaatsleutels *samengesteld* zijn en de kandidaatsleutels *overlappend* zijn (d.w.z. tenminste één attribuut gemeenschappelijk hebben). Nemen we als voorbeeld onderstaande tabel met de attributen studentnummer, vak en staflid.

Een student kan meerdere vakken kiezen, en per vak kan één of meer leden

student	vak	staflid
100	wiskunde	Cauchy
150	psychologie	Jung
200	wiskunde	Riemann
250	wiskunde	Cauchy
300	psychologie	Freud
300	wiskunde	Riemann

van de onderwijsstaf als begeleider optreden. Een staflid is evenwel slechts begeleider van één vak. Voor bovenstaande tabel geldt dat er twee kandidaatsleutels zijn : de combinatie (student, vak) enerzijds en de combinatie (student, staflid) anderzijds. Bovendien geldt voor bovenstaande tabel dat het attribuut staflid bepalend is voor vak. Met behulp van de naam van het staflid kunnen we m.a.w. direct het vak bepalen. De relatie heeft de eerste normaalvorm, de tweede normaalvorm (welke van de twee kandidaatsleutels we ook kiezen, alle niet-sleutelattributen zijn volledig afhankelijk van de gehele sleutel) en de derde normaalvorm (geen transitieve afhankelijkheden). Toch is de relatie onderhevig aan anomalieën. Indien student 300 verwijderd wordt, verliezen we de informatie dat staflid Freud begeleider is voor het vak psychologie, een verwijderanomalie. Er is eveneens sprake van een invoeganomalie. We kunnen pas registreren dat staflid Keynes begeleider is voor het vak economie indien een student dit als één van de vakken kiest. In bovenstaande relatie hebben we twee overlappende kandidaatsleutels. Daarbij is het attribuut 'staflid' alleen geen kandidaatsleutel omdat deze het attribuut 'student' niet bepaalt. Om die reden is de relatie niet in BCNV, immers het attribuut 'staflid' is determinant (van het attribuut 'vak') en geen kandidaatsleutel.

Vandaar dat we de relatie zoals hieronder opsplitsen in twee nieuwe relaties. De Boyce-Codd normaalvorm definiëren we dan ook als volgt : *een relatie is in Boyce-Codd normaalvorm indien iedere determinant ook een kandidaatsleutel is.*

Relaties in BCNV bevatten geen anomalieën meer die veroorzaakt worden door functionele afhankelijkheden. Het leek er even op dat hiermee het probleem was opgelost. Al spoedig werd echter ontdekt dat ook in situaties zonder functionele afhankelijkheid anomalieën kunnen optreden.

student	staflid
100	Cauchy
150	Jung
200	Riemann
250	Cauchy
300	Freud
300	Riemann

sleutel = (student,staflid)

staflid	vak
Cauchy	wiskunde
Jung	psychologie
Riemann	wiskunde
Freud	psychologie

sleutel = staflid

4.3.2.1.5. De vierde normaalvorm

Regel 5 - Een relatie is in vierde normaalvorm als de relatie in BCNV staat en geen meerwaardige afhankelijkheden bevat.

In onderstaande tabel wordt een relatie voorgesteld met gegevens over studenten, vakken en activiteiten. De relatie is in 2NV omdat het gehele tupel de sleutel is, in 3 NV omdat er geen transitieve afhankelijkheden in voorkomen en in BCNV omdat er geen niet-sleutel-determinanten in voorkomen. Nochtans moet de relatie genormaliseerd worden.

studentnummer	vak	activiteit
100	muziek	zwemmen
100	boekhouden	zwemmen
100	muziek	tennis
100	boekhouden	tennis
150	wiskunde	joggen

sleutel = (studentnummer,vak,activiteit)

Een student kan meerdere vakken kiezen en kan zich opgeven voor verschillende activiteiten. Dit betekent dat alleen de combinatie (studentnummer, vak, activiteit) in aanmerking komt als sleutel. Een relatie als onderstaande wordt benoemd als *meerwaardige afhankelijkheid* ("multivalued dependence"). In het algemeen is er sprake van meerwaardige afhankelijkheid als er in een relatie minstens drie attributen voorkomen, waarvan er twee bij dezelfde waarde van een derde, verschillende waarden kunnen aannemen. In het geval van meervoudige afhankelijkheid kunnen

modificatie-anomalieën optreden. We specifiëren dit nader. Het lijkt erop alsof student 100 alleen zwemt in de functie van muziekleerling en alleen tennist in de functie van boekhoudleerling, terwijl in werkelijkheid vakken en activiteiten helemaal niets met elkaar te maken hebben. Veronderstel dat student 100 ook gaat skiën. We zouden een rij (100, muziek, skiën) kunnen toevoegen aan de tabel en dan lijkt het erop dat student 100 enkel gaat skiën in de functie van muziekleerling. Om dit te vermijden zouden we kunnen een tupel (100, boekhouden, skiën) toevoegen. Voor een kleine wijziging moet dus een groot aantal tupels aan de tabel toegevoegd worden. Vandaar ook de normalisering zoals hieronder voorgesteld.

studentnummer	vak
100	muziek
100	boekhouden
150	wiskunde

studentnummer	activiteit
100	zwemmen
100	tennis
100	skiën
150	joggen

4.3.2.1.6. De domein/sleutel-normaalvorm

Regel 6 - *Een relatie is in DS/NV als iedere randvoorwaarde bij de relatie een logisch gevolg is van de definitie van de sleutels en de domeinen*.

In 1981 werd een belangrijk artikel van de hand van R. Fagin gepubliceerd, waarin de *domein/sleutel-normaalvorm* (DS/NV) werd gedefinieerd. Voor zover het modificatie-anomalieën betreft, is hiermee een eind gekomen aan het zoeken naar hogere normaalvormen.

De DS/NV maakt alleen gebruik van de begrippen *sleutel* en *domein*. Dit zijn kernbegrippen bij het werken met databases en komen voor in (bijna) ieder DBMS-product. In feite gaf het werk van Fagin een formele basis en een rechtvaardiging aan datgenen wat velen in de praktijk al intuïtief inzagen, maar zelf niet helder konden formuleren.

Het idee achter de DS/NV is verrassend eenvoudig. De belangrijke termen in deze definitie zijn : *randvoorwaarde, sleutel en domein.*

Het begrip 'randvoorwaarde' wordt hier in de ruime zin opgevat. Dus alle regels over toegelaten waarden, functionele afhankelijkheden en meerwaardige afhankelijkheden zijn randvoorwaarden. De tweede

belangrijke term sleutel definieerden we al eerder als een grootheid die een tupel eenduidig kan identificeren. De derde belangrijke term in de definitie is domein. Een domein is de beschrijving van de toegelaten waarden van een attribuut.

Informeel is een relatie in domein/sleutel-normaalvorm als aan alle randvoorwaarden is voldaan en als we ons houden aan de beperkingen die aan sleutels en domeinen zijn opgelegd. Tot op heden is er evenwel geen formele methode gevonden om een relatie in DS/NV om te zetten. Het zoeken naar of ontwerpen van DS/NV-relaties is op dit moment meer een kunst dan een wetenschap. Om die reden kan men de DS/NV beschouwen als een ontwerpdoelstelling. We streven ernaar relaties zodanig te definiëren dat randvoorwaarden een logisch gevolg zijn van de gekozen domeinen en sleutels. We zullen de DS/NV verduidelijken aan de hand van een tweetal voorbeelden.

4.3.2.1.6.1. DS/NV : voorbeeld 1

Veronderstellen we een relatie met als attributen studentnummer, studiefase, (studenten)huis en huur. Door studentnummer (SNR) zijn de drie andere attributen functioneel bepaald zodat we SNR als sleutel kiezen. Anderzijds is er de specificatie dat de te bepalen huur functioneel bepaald wordt door (studenten)huis. Een andere randvoorwaarde is dat het studentnummer niet mag beginnnen met een 1. Als we deze beide randvoorwaarden kunnen uitdrukken als logisch gevolg van de domein- en sleuteldefinities kunnen we er volgens Fagin zeker van zijn dat de relaties geen modificatie-anomalieën bevatten.

We kunnen er ons eenvoudig van verzekeren dat studentnummers niet met een 1 beginnen door dit in de domeinbeschrijving op te nemen. Vervolgens moeten we er ons van verzekeren dat de functionele afhankelijkheid huis => huur een logisch gevolg is van de gekozen sleutels. Dit zou al het geval zijn als huis zelf een sleutel was, en dus is het de vraag hoe we dit kunnen realiseren. In de relatie student kan huis geen sleutel zijn omdat meer dan één student in hetzelfde huis kan wonen, maar misschien zou huis een sleutel kunnen zijn in een afzonderlijke relatie. Om die reden wordt een relatie gedefinieerd met de attributen huis en huur waarbij huis als sleutel gedefinieerd wordt. De attributen huis en huur worden derhalve uit de oorspronkelijke relatie verwijderd.

Samenvattend kunnen volgende domeindefinities en relatie- en sleuteldefinities vooropgesteld worden.

domeindefinities

SNR IN CDDD (C=decimaal cijfer <> 1 en DDD is decimaal cijfer)
FASE IN ("1KAN", "2KAN", "1LIC", "2LIC", "3LIC")
HUIS IN CH(25)
HUUR IN N(5)

relatie- en sleuteldefinities

relatie STUDENT (SNR, FASE, HUIS)
sleutel = SNR

relatie HUIS (huis, huur)
sleutel = huis

4.3.2.1.6.2. DS/NV : voorbeeld 2

Een wat ingewikkelder voorbeeld dan het vorige is een relatie die gegevens bevat over docenten, de vakken die ze geven en de studenten die zij begeleiden. We veronderstellen dat docentnummer (DNR) en docentnaam (DNAAM) afzonderlijk een docent identificeren (de docentnamen zijn dus zo geformuleerd dat ze niet dubbel kunnen voorkomen). Studentnummer (SNR) is de unieke identificatie van een student en elke student kan slechts door één docent begeleid worden. Een docent kan meer vakken geven en kan ook begeleider zijn van meer studenten. DNR's beginnen steeds met een 1, SNR's juist niet.

In een dergelijke wat ingewikkelder situatie is een meer intuïtieve benadering op zijn plaats. Vandaar dat we de oorspronkelijke relatie met de vijf attributen zouden kunnen voorstellen in onderstaande tabel.

docentnummer	docentnaam	vak	studentnummer	studentnaam
100	Janssens	wiskunde	201	Vertommen
100	Janssens	wiskunde	301	Leys
100	Janssens	boekhouden	401	De Paepe
200	Pieters	fysica	501	Brebels
200	Pieters	aardrijkskunde	601	Heymans
300	Casteels	informatica	701	Vertommen
300	Casteels	informatica	801	Peersman

Bovenstaande tabel is niet in 2NV omdat de niet-sleutelattributen niet afhankelijk zijn van de gehele sleutel. Er zal dus moeten opgesplitst worden in afzonderlijke tabellen. Als we de attributen docentnummer, docentnaam en vak in een eerste tabel onderbrengen, dan is deze relatie evenmin in 2NV omdat het niet-sleutelattribuut vaak niet afhankelijk is van de gehele sleutel (docentnummer, docentnaam). Vandaar dat we deze relatie opsplitsen in twee relaties, nl. de relatie (docentnummer, docentnaam) en de relatie (docentnaam, vak). Aan deze twee relaties wordt tenslotte nog een derde relatie toegevoegd met als attributen studentnummer, studentnaam en vak. In deze relatie zijn de attributen studentnummer en studentnaam de primaire sleutel. Het attribuut vak kan in deze tabel opgenomen worden omdat dit attribuut afhankelijk is van de gehele sleutel.

4.3.2.2. CASE-tools

De laatste jaren is het maken van gegevensmodellen aanmerkelijk gemakkelijker geworden, omdat veel zogenaamde CASE-tools (*Computer Aided Software Engeneering*) het opstellen van diagrammen ondersteunen. In hetgeen volgt gaan we wat nader in op het gebruik van het *entiteit-relatiemodel*. Het entiteit-relatiemodel (E-R-model) werd voor het eerst in 1976 beschreven door Peter Chen[7]. Sindsdien is het uitgebreid en gewijzigd door Chen zelf en door anderen. Het *E-R-model* wordt inmiddels gebruikt als onderdeel van een aantal CASE-tools en ook hierdoor heeft het verdere wijzigingen ondergaan. Er is op dit moment geen sprake van één standaard E-R-model, maar er bestaan een aantal gemeenschappelijke elementen die in de meeste varianten van het E-R-model terug te vinden zijn. We zullen deze gemeenschappelijke elementen in hetgeen volgt beschrijven.

Een *entiteit* is een grootheid uit de werkomgeving van een gebruiker. Het is iets dat betekenis heeft voor de gebruiker binnen de context van een te bouwen systeem. Voorbeelden van entiteiten zijn KLANT, MEDEWERKER, PRODUCT, e.d. Een instantie van een entiteit is een verwijzing naar een bestaande entiteit, bijvoorbeeld KLANT 1234.

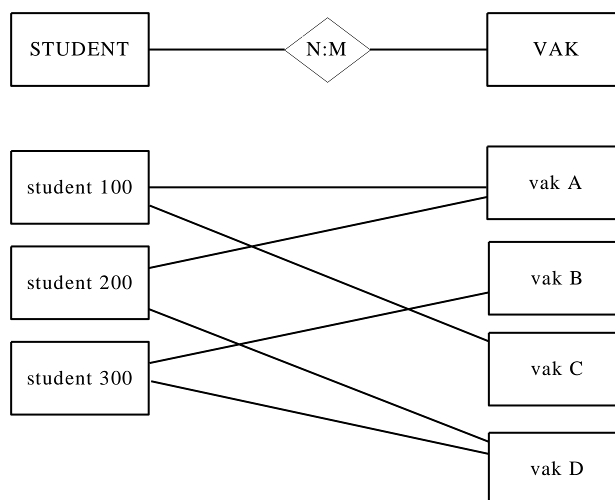
Entiteiten hebben eigenschappen of *attributen* die bijzonderheden van de entiteit aangeven. Voorbeelden zijn klantnummer, klantnaam, klantadres, e.d. Entiteitinstanties worden geïdentificeerd aan de hand van attributen. Om een bepaalde instantie te kunnen identificeren, moet er een attribuut bestaan dat per instantie een unieke waarde aanneemt.

Er kan verband bestaan tussen entiteiten. Dergelijke verbanden worden relaties of relationele verbanden genoemd. Er kunnen relaties bestaan tussen

2 of meer entiteiten. Het aantal entiteiten in een relatie wordt de graad van die relatie genoemd. Meestal gaat het om relaties tussen 2 entiteiten. Deze relaties worden *binaire relaties* genoemd. In een 1 : 1 of *één-op-één* relatie is steeds een enkele instantie van de ene entiteit verbonden met een enkele instantie van een andere entiteit. In een 1 : N of *één-op-veel* relatie is elke instantie van de ene entiteit verbonden met meerdere instanties van een andere entiteit. In het omgekeerde geval wordt gesproken van een veel-op-één relatie (N : 1-relatie). Ten slotte bestaan er ook N : M of *veel-op-veel* relaties.

In entiteit-relatiediagrammen, kortweg *E-R-diagrammen* genoemd, worden entiteiten weergegeven door *rechthoeken* en de relaties tussen entiteiten door *ruiten*. In die ruiten wordt de *maximale cardialiteit* aangegeven. De maximale cardialiteit is het maximum aantal entiteiten dat met een andere entiteit kan verbonden zijn. De minimale cardialiteit wordt niet aangegeven in E-R-diagrammen. De minimale cardialiteit kan worden aangegeven door middel van een rechtopstaand streepje als een entiteit moet bestaan en door een ovaaltje of een nul als een entiteit niet hoeft te bestaan. In sommige versies van E-R-diagrammen worden de attributen van entiteiten in ellipsen geschreven en verbonden met de bijhorende entiteit. Bij veel toepassingen kunnen lang niet alle attributen op deze wijze in een E-R-diagram worden weergegeven. Het zijn er teveel en het diagram wordt onoverzichtelijk. De entiteitattributen kunnen dan in een afzonderlijk overzicht opgesomd worden.

Voorbeeld van een E-R-diagram met een veel-op-veel relatie



De weergave van entiteiten met behulp van het relationele model is in het algemeen niet moeilijk. Per entiteit wordt een relatietabel gedefinieerd. Deze tabel krijgt dezelfde naam als de entiteit en de attributen van de entiteit worden gebruikt als attributen van de relatie. Vervolgens worden op de tabel de besproken normalisatieregels toegepast waardoor in sommige gevallen veranderingen moeten aangebracht worden, bijvoorbeeld wanneer er modificatie-anomalieën bestaan.

In het voorbeeld hieronder worden voor de entiteit KLANT volgende attributen gedefinieerd : klantnummer, klantnaam, adres, stad, land, postcode, contactnaam en telefoonnummer. Deze attributen worden overgenomen als attributen binnen de relatie KLANT. Als we welk attribuut de entiteit identificeert, definiëren we dit attribuut als de sleutel in de relatie. In dit geval veronderstellen we dat klantnummer de sleutel is.

Eens we de attributen van de entiteit omschreven hebben en de entiteit omgezet hebben in een relatie, wordt de relatie geanalyseerd aan de hand van de normalisatiecriteria. De vraag die gesteld wordt is of de relatie KLANT in domein/sleutel normaalvorm staat. Om dat te bepalen, moeten we weten wat de randvoorwaarden bij de relatie zijn. Zolang we geen volledige beschrijving hebben van de attributen van de entiteit KLANT en we niet alle beperkingen kennen die aan domeinen van attributen opgelegd zijn, weten we niet precies welke de randvoorwaarden zijn. Een aantal randvoorwaarden kunnen we evenwel afleiden uit de namen van de attributen.

De waarde van klantnummer bepaalt de waarde van alle andere attributen. Dit betekent dat de waarden van klantnaam, adres, stad, land, postcode, contactnaam en telefoonnummer bekend zijn, van zodra we de waarde van klantnummer kennen.

Er zijn nog andere randvoorwaarden die volgen uit functionele afhankelijkheden tussen attributen. Door postcode bijvoorbeeld wordt stad en land bepaald (mogelijk ook door adres, maar dat laten we hier buiten beschouwing). Verder bepaalt contactnaam ook telefoonnummer. Om relaties in domein/sleutel normaalvorm te creëren, moeten we deze functionele afhankelijkheden een logisch gevolg maken van de keuze van de domeinen en sleutels. Vandaar dat we drie relaties definiëren, zoals hieronder aangegeven.

KLANT (<u>klantnummer</u>, klantnaam, adres, postcode, contactnaam)

POSTCD (<u>postcode</u>, stad, land)

CONTACT (<u>contactnaam</u>, telefoonnummer)

randvoorwaarden :
⇒ postcode in KLANT moet voorkomen in postcode in POSTCD
⇒ contactnaam in KLANT moet voorkomen in contactnaam in CONTACT

Door bovenstaand ontwerp hoeven we ons geen zorgen te maken over verdere modificatie-anomalieën. Nieuwe postcodes en contacten kunnen worden toegevoegd, zonder dat hiervoor eerst een klant moet worden toegevoegd met deze postcode of dit contact. Als we een klant met een gegeven postcode verwijderen, heeft dit evenmin tot gevolg dat we de informatie bij deze postcode (stad en land) kwijt zijn.

Toch zullen we vaak niet kiezen voor deze oplossing. In een dergelijk eenvoudig geval is het meestal niet zinvol de oorspronkelijke tabel in drie stukken te hakken. In de praktijk zullen we waarschijnlijk kiezen voor de oorspronkelijke tabel KLANT. De hiermee verbonden modificatie-anomalieën zijn wellicht niet zo belangrijk. Wie heeft er belang bij het telefoonnummer van contactpersoon Janssens te kennen als het bedrijf waar Janssens werkt geen klant meer is ?

Samenvattend. Als we een entiteit met het relationele model willen weergeven, is de eerste stap de constructie van een relatie die alle eigenschappen van de entiteit als attributen heeft. Vervolgens beoordelen we deze relatie volgens de normalisatiecriteria en in veel gevallen kan het ontwerp verbeterd worden door de relatie om te zetten in twee of meer relaties in domein/sleutel-normaalvorm. Als het resultaat echter te geconstrueerd, onnodig verbrokkeld en/of moeilijk werkbaar lijkt, kan voor een niet DS/NV-oplossing worden gekozen. Als twee of drie relaties moeten benaderd worden om alle gegevens over een klant te krijgen, kan de responstijd wel eens onacceptabel lang zijn. Belangrijk is evenwel het volgende voor ogen te houden. Of we nu wel of niet beslissen te normaliseren, altijd moeten we alle relaties beoordelen aan de hand van de normalisatiecriteria. Als we al gaan zondigen tegen de normalisatieregels, moeten we in ieder geval precies weten wat daarvan de gevolgen zijn en welke modificatie-anomalieën in de door ons gekozen oplossing verborgen zitten.

4.4. Basisbeginselen bij de opzet van databases

Het implementeren van een relationele database gebeurt in een aantal stappen. Eerst moet de structuur van de database voor het DBMS worden gedefinieerd. Hiervoor wordt een gegevensdefinitietaal (*Data Definition Language* of DDL) gebruikt, of een daarmee equivalente methode zoals een grafische weergave. Daarna wordt ruimte gereserveerd op de te gebruiken fysieke opslagmedia (bijvoorbeeld een harde schijf) en wordt de database gevuld met gegevens.

4.4.1. Het definiëren van de databasestructuur in het DBMS

De structuur van een database kan afhankelijk van het gebruikte systeem, op verschillende manieren voor een DBMS worden beschreven. Bij sommige systemen moet een *tekstbestand* worden opgesteld dat een beschrijving geeft van de databasestructuur. De taal die daarbij gebruikt wordt, wordt een DDL genoemd. In de DDL-file worden de tabellen van de database opgesomd, per tabel worden de kolommen opgesomd, hun types, eventuele indexen en randvoorwaarden (validatieregels). Bij andere databasesystemen hoeft de database niet behulp van een DDL in een tekstbestand te worden gedefinieerd. Zij maken gebruik van een *grafische weergave* van de databasestructuur. Over het algemeen zijn grafische definitiehulpmidelen gemakkelijker te gebruiken dan op tekst gebaseerde DDL's. In het geval van een grafische weergave dient een ontwikkelaar niet te onthouden hoe lang een veldnaam maximaal mag zijn ; op het scherm is een vast aantal posities voor veldnamen gereserveerd. Als men een tekstbestand gebruikt, dan blijkt pas bij een poging tot creëren van de databasestructuur uit een foutmelding dat ergens een te lange veldnaam gebruikt is.

Hoe de databasestructuur ook wordt gedefinieerd, in ieder geval moeten de tabellen worden benoemd en moeten de formaten van de attributen worden aangegeven. Afhankelijk van de mogelijkheden van het systeem kunnen soms ook randvoorwaarden worden gespecifieerd die het DBMS moet controleren. Zo kunnen kolomwaarden bijvoorbeeld worden gedefinieerd als NOT NULL (cfr. infra). Soms ook kan het waardebereik van attributen worden gespecifieerd.

In SQL wordt voor het opzetten van tabellen de CREATE TABLE-instructie gebruikt. In een tabelschema worden de naam van de tabel, de kolomdefinities en diverse integriteitsregels opgenomen. Hieronder wordt

een voorbeeld gegeven van het opzetten van een tabel, SPELERS genaamd, met de *CREATE TABLE-instructie*.

```
CREATE TABLE SPELERS (
     SPELERSNR  SMALLINT   NOT NULL   ,
     NAAM       CHAR(15)   NOT NULL   ,
     VOORNM     CHAR(15)   NOT NULL   ,
     GEB_JR     SMALLINT              ,
     STRAAT     CHAR(15)   NOT NULL   ,
     HUISNR     CHAR(4)               ,
     POSTCODE   CHAR(4)               ,
     PLAATS     CHAR(15)   NOT NULL   ,
     TEL        CHAR(10)
     PRIMARY KEY (SPELERSNR)          )
```

De naam van de tabel is het eerste dat gespecifieerd wordt in het CREATE TABLE-commando. De lijst met de kolommen waaruit de tabel bestaat, staat tussen haakjes. Achter elke kolom staat het datatype, dat aangeeft welk soort waarde in die betreffende kolom ingevoerd mag worden, eventueel aangevuld met de NOT NULL-specificatie.

Kolommen worden gevuld met waarden. Een waarde kan bijvoorbeeld een getal zijn, een woord of een datum. Een speciaal soort waarde is de *NULL-waarde*. De NULL-waarde is te vergelijken met 'waarde onbekend' of 'waarde niet aanwezig'. NULL-waarden zijn niet hetzelfde als het getal nul of een rij spaties. Een NULL-waarde heeft de eigenschap dat zij nooit gelijk is aan een andere NULL-waarde. Twee NULL-waarden zijn dus niet gelijk aan elkaar, maar ook niet ongelijk. Als we wel zouden weten dat twee NULL-waarden gelijk of ongelijk zijn, zouden we 'iets' van die NULL-waarden afweten. We zouden dan niet meer kunnen zeggen dat de twee waarden (totaal) onbekend zijn. In een CREATE TABLE-instructie mag achter het datatype van een kolom de zogenaamde *NOT NULL-integriteitsregel*, ook wel NOT NULL-optie genoemd, gespecifieerd worden. Hiermee wordt aangegeven welke kolommen geen NULL-waarden mogen bevatten. Met andere woorden : in alle rijen moet elke NOT NULL-kolom gevuld worden met een waarde.

Aangezien een NULL-waarde nooit gelijk is aan een andere NULL-waarde, kunnen twee tabellen nooit gerelateerd worden op NULL-waarden. NULL-waarden worden evenmin meegenomen in totaalberekeningen. Een waarde kan gecontroleerd worden om te achterhalen of die een NULL bevat door de waarde te vergelijken met het speciale sleutelwoord NULL of de ingebouwde functie *IsNULL* te gebruiken.

In Microsoft Access voor Windows 95 kan het onderscheid gemaakt worden tussen *NULL-waarden* en *tekst met een lengte van nul*. Met dit laatste wordt aangegeven dat de waarde in een veld bekend is maar het veld leeg is. Tabellen kunnen gerelateerd worden op de vergelijking van tekst met een lengte van nul en twee teksten met een lengte van nul worden als gelijk beschouwd bij een vergelijking. Het is evenwel nodig bij tekstvelden eerst de eigenschap ‘lengte nul’ toe te staan om gebruikers toe te laten tekst met een lengte van nul in te voeren. Wordt dat niet gedaan dan converteert Access alle tekst met een lengte van nul en lege tekst naar een NULL-waarde.

Waarom is het belangrijk een onderscheid te maken tussen NULL-waarden en tekst met een lengte van nul ? We verduidelijken dit aan de hand van een voorbeeld. Veronderstel dat we een tabel hebben waarin de resultaten worden opgeslagen van een onderzoek naar de voorkeuren voor auto’s. Bij enquêteformulieren waarbij de vraag naar de voorkeur voor een kleur is ingevuld, moet dan een NULL-waarde gebruikt worden om aan te geven dat het gegeven ontbreekt. We willen immers vermijden dat bij het analyseren van de antwoorden immers een ‘onbekend’ antwoord niet relateren aan een andere ‘onbekend’ antwoord en we willen deze antwoorden ook niet verwerken bij het berekenen van bijvoorbeeld gemiddelden, e.d. Sommige mensen zullen echter op de vraag naar hun kleurvoorkeur antwoorden met ‘het kan me niet schelen’ . In dit geval hebben we te maken met een bekend, leeg antwoord en is tekst met een lengte van nul van toepassing. Deze antwoorden kunnen derhalve meegenomen worden bij berekeningen. Een andere voorbeeld is dat van de opslag van faxnummers in een tabel. Slaat men een NULL-waarde op, dan betekent dit dat we niet weten of iemand een faxnummer heeft. Slaan we tekst met een waarde van nul op, dan weten we dat het faxnummer ontbreekt.

In het CREATE TABLE-commando wordt eveneens de *primaire sleutel* van de tabel vermeld. Een primaire sleutel van een tabel is een kolom (of een combinatie van kolommen) waarvoor geldt dat elke waarde maximaal eenmaal mag voorkomen. Met het definiëren van de primaire sleutel in de SPELERS-tabel wordt dus aangegeven dat elke spelersnummer maximaal eenmaal binnen de SPELERSNR-kolom mag voorkomen. *Een primaire sleutel is een bepaald soort integriteitsregel.* In SQL worden primaire sleutels gespecifieerd binnen een CREATE TABLE-instructie met de woorden PRIMARY KEY. Achter PRIMARY KEY worden tussen haakjes de namen van de kolommen waaruit de primaire sleutel is opgebouwd, aangegeven.

4.4.1.1. Primaire sleutels

Een primaire sleutel is een kolom of een verzameling kolommen van een tabel waarvan de waarden te allen tijde uniek zijn. NULL-waarden zijn niet toegestaan in de kolommen die onderdeel zijn van een primaire sleutel. Primaire sleutels mogen over meerdere kolommen van een tabel gedefinieerd worden. We noemen dit *samengestelde primaire sleutels* (“composite primary key”).

Er zijn een aantal regels waaraan de kolommen van een primaire sleutel behoren te voldoen. Sommige van deze regels stammen uit de theorie van het relationele model en andere worden door SQL opgelegd. Het is aan te raden bij het definiëren van een primaire sleutel onderstaande regels in acht te nemen.

1. Voor elke tabel kan maximaal één primaire sleutel gedefinieerd worden.

2. De theorie van het relationele model vereist dat voor elke tabel minimaal één primaire steutel gedefinieerd wordt. SQL dwingt dit echter niet af. Het is een goede gewoonte voor elke basistabel een primaire sleutel te definiëren.

3. Twee verschillende rijen in een tabel mogen nooit dezelfde waarde voor de primaire sleutel hebben. Dit wordt de *uniciteitsregel* genoemd.

4. Een primaire steutel is incorrect indien het mogelijk is een kolom uit de primaire sleutel te verwijderen en deze ‘smallere’ primaire sleutel blijft voldoen aan de uniciteitsregel. Deze regel wordt de *minimaliteitsregel* genoemd. In het kort wil dit zeggen dat een primaire sleutel niet onnodig veel kolommen behoort te bevatten. Een te brede primaire sleutel voldoet niet aan de minimaliteitsregel.
5. Een kolomnaam mag maar éénmaal voorkomen binnen de kolommenlijst van een primaire sleutel.

6. De kolommen behorende tot een primaire sleutel mogen geen NULL-waarden bevatten. Deze regel wordt de *entiteit-integriteitsregel* genoemd.

4.4.1.2. Alternatieve sleutels

In het relationele model is een alternatieve sleutel, net als een primaire sleutel, een kolom of een verzameling kolommen van een tabel waarvan de waarden te allen tijde uniek zijn. Een alternatieve sleutel is een kandidaat-sleutel die niet tot de primaire sleutel verkozen is. Er bestaan twee belangrijke verschillen tussen primaire en alternatieve sleutels. Ten eerste, primaire sleutels mogen geen NULL-waarden bevatten, alternatieve sleutels wel (tenzij dit expliciet met een NOT NULL-integriteitsregel wordt verboden). Ten tweede, voor een tabel mag slechts één primaire, maar mogen meerdere alternatieve sleutels gedefinieerd worden.

In onderstaand voorbeeld wordt de SPELERSNR-kolom in de tabel WEDSTRDN gedefinieerd als een alternatieve sleutel en wordt aangegeven dat de waarden uniek moeten blijven :

```
CREATE TABLE WEDSTRDN(
   WEDSTRNR   SMALLINT   NOT NULL,
   SPELERSNR  SMALLINT   NOT NULL UNIQUE
   PRIMARY KEY(WEDSTRNR))
```

4.4.1.3. Refererende sleutels

Een refererende sleutel (ook vreemde sleutel of "foreign key" genoemd) is een kolom of een combinatie van kolommen waarvan de inhoud verwijst naar de overeenkomstige waarde van een primaire sleutel is een andere tabel. Op die manier worden tabellen logisch gekoppeld. Regels die betrekking hebben op de relaties tussen tabellen onderling worden *refererende integriteitsregels* genoemd. Refererende integriteitsregels zijn een speciaal soort integriteitsregel die in een CREATE TABLE-instructie kunnen geïmplementeerd worden.

De tabel waarin een refererende sleutel gedefinieerd wordt, noemen we de refererende tabel. De tabel waarnaar een refererende sleutel verwijst, noemen we de gerefereerde tabel. Wat voor effect heeft het definiëren van een refererende sleutel eigenlijk ? Na het definiëren van een refererende sleutel zal SQL garanderen dat elke niet NULL-waarde, die in de refererende sleutel ingevoerd wordt, reeds voorkomt in de primaire sleutel van de gerefereerde tabel. Nemen we het eerder gegeven voorbeeld van de SPELERS-tabel waarin de kolom SPELERSNR als primaire sleutel

gedefinieerd werd. Met een nieuwe CREATE TABLE-instructie waarbij een TEAMS-tabel gecreëerd wordt, definiëren we een refererende sleutel die verwijst naar de primaire sleutel in de SPELERS-tabel :

```
CREATE TABLE TEAMS (
   TEAMNR     SMALLINT   NOT NULL,
   SPELERSNR  SMALLINT   NOT NULL,
   DIVISIE    CHAR(6)    NOT NULL
   PRIMARY KEY(TEAMNR)
   FOREIGN KEY(SPELERSNR)
   REFERENCES SPELERS)
```

Aan de CREATE TABLE-instructie is de specificatie voor een refererende sleutel toegevoegd. Elke specificatie van een refererende sleutel bestaat uit drie delen : de kolom die de refererende sleutel is (FOREIGN KEY), de gerefereerde tabel (REFERENCES) en de refererende actie (cfr. infra).

Bij het tweede deel rijst direct de vraag : naar welke kolom verwijst de refererende sleutel ? *Refererende sleutels kunnen alleen naar primaire sleutels verwijzen*. In het gegeven voorbeeld verwijst de refererende sleutel naar de primaire sleutel uit de SPELERS-tabel, nl. SPELERSNR. We hadden het commando ook als volgt mogen schrijven :

```
CREATE TABLE TEAMS (
   TEAMNR     SMALLINT   NOT NULL,
   SPELERSNR  SMALLINT   NOT NULL,
   DIVISIE    CHAR(6)    NOT NULL
   PRIMARY KEY(TEAMNR)
   FOREIGN KEY(SPELERSNR)
   REFERENCES SPELERS (SPELERSNR))
```

De kolomnamen van de primaire sleutel (in de gerefereerde tabel) mogen weggelaten worden indien de naam van de refererende sleutel gelijk is aan de naam van primaire sleutel. Het definiëren van een refererende sleutel heeft, zoals eerder vermeld, tot gevolg dat SQL nagaat of elke niet NULL-waarde die ingevuld wordt in de kolom SPELERSNR in de TEAMS-tabel reeds bestaat in de kolom SPELERSNR van de SPELERS-tabel. Indien dit niet het geval is, ontvangt de gebruiker of de applicatie een foutmelding en wordt de mutatie afgekeurd. Het definiëren van een refererende sleutel zoals in bovenstaand voorbeeld heeft derhalve tot gevolg dat onderstaande SELECT-instructie nooit een resultaat heeft :

```
SELECT   *
FROM     TEAMS
WHERE    SPELERSNR  NOT  IN
         (SELECT SPELERSNR FROM SPELERS)
```

Uit de SPELERS-tabel kan slechts een speler verwijderd worden voor zover de waarde ervan in de SPELERSNR-kolom van de TEAMS-tabel niet gekend is. Eveneens geldt dat het wijzigen van de waarde in de SPELERSNR-kolom van de tabel SPELERS slechts doorgevoerd kan worden voor zover dit nummer niet gekend is in de TEAMS-tabel. Voor het invoeren van nieuwe spelers in de SPELERS-tabel worden door de refererende sleutel geen beperkingen opgelegd.

Voor het specifiëren van een refererende sleutel gelden de volgende regels.

1. De gerefereerde tabel moet een bestaande tabel zijn die eerder met een CREATE TABLE-instructie gecreëerd is of moet een tabel zijn die gecreëerd wordt. De refererende tabel is dan dezelfde tabel als de gerefereerde tabel. Een dergelijke tabel wordt een "self referencing table" genoemd en de constructie zelf wordt "self referencing integrity" genoemd.

2. Voor de gerefereerde tabel moet een primaire sleutel gedefinieerd zijn.

3. Achter de tabelnaam waarnaar gerefereerd wordt, mag een kolomnaam (of een verzameling kolomnamen) gespecifieerd worden. Als dit gedaan wordt, moet deze kolom de primaire sleutel van de tabel zijn. Indien er geen kolomnaam gespecifieerd wordt, moet de gerefereerde tabel een primaire sleutel hebben en moet de kolomnaam van de refererende sleutel gelijk zijn aan die van de primaire sleutel in de gerefereerde tabel.

4. Een NULL-waarde in een refererende sleutel is toegestaan, ook al kan een primaire sleutel zelf nooit NULL-waarden bevatten. Dit betekent dat de inhoud van een refererende sleutel correct is indien elke niet NULL-waarde voorkomt in de primaire sleutel van de gerefereerde tabel.

5. Een refererende sleutel mag uit één of meer kolommen bestaan. Dit betekent dat als een refererende sleutel uit bijvoorbeeld twee kolommen bestaat, de primaire sleutel van de gerefereerde tabel ook minstens uit twee kolommen moet bestaan. Een deelverzameling kolommen van een primaire sleutel, mag een refererende sleutel vormen.

6. De datatypes van de kolommen van de refererende sleutel moeten gelijk zijn aan die van de kolommen van de primaire sleutel van de gerefereerde tabel.

Eén onderdeel van de refererende sleutel hebben tot nu toe overgeslagen : de *refererende actie*. Tot nog toe zijn we er van uitgegaan dat bijvoorbeeld een record in de SPELERS-tabel niet kan verwijderd worden indien ditzelfde record ook aanwezig is in de kolom SPELERSNR van de TEAMS-tabel. Deze beveiliging komt er door een refererende sleutel te definiëren. Door een refererende actie kunnen we dit veranderen.

Bij elke refererende sleutel kunnen *twee refererende acties* gedefinieerd worden. Het niet-specifiëren van refererende acties is gelijkwaardig aan het specifiëren van de volgende twee refererende acties :

```
ON UPDATE RESTRICT
ON DELETE RESTRICT
```

Met de eerste refererende actie wordt expliciet gespecifieerd dat als het nummer van een speler (in de refererende tabel) gewijzigd wordt (UPDATE), de wijziging afgekeurd (RESTRICT) moet worden. Hetzelfde geldt voor de tweede refererende actie : als een speler in de refererende tabel verwijderd wordt (DELETE), dan moet de verwijdering afgekeurd worden (RESTRICT).

Een tweede mogelijkheid is CASCADE i.p.v. RESTRICT :

```
CREATE TABLE TEAMS (
   TEAMNR     SMALLINT   NOT NULL,
   SPELERSNR  SMALLINT   NOT NULL,
   DIVISIE    CHAR(6)    NOT NULL
   PRIMARY KEY(TEAMNR)
   FOREIGN KEY(SPELERSNR)
   REFERENCES SPELERS
   ON UPDATE CASCADE
   ON DELETE CASCADE)
```

Als het spelersnummer van een speler in de SPELERS-tabel gewijzigd wordt, wordt in de TEAMS-tabel het spelersnummer eveneens veranderd. Veronderstel dat de volgende UPDATE-instructie gegeven wordt :

```
UPDATE   SPELERS
SET      SPELERSNR = 80
WHERE    SPELERSNR = 127
```

SQL zal automatisch de volgende UPDATE-instructie (achter de schermen, CASCADE) uitvoeren :

```
UPDATE   TEAMS
SET      SPELERSNR = 80
WHERE    SPELERSNR = 127
```

Hetzelfde geldt voor het verwijderen van spelers.

Als we het woord CASCADE veranderen door SET NULL, de derde mogelijkheid, krijgen we weer een andere reactie.

```
CREATE TABLE TEAMS (
   TEAMNR      SMALLINT   NOT NULL,
   SPELERSNR   SMALLINT   NOT NULL,
   DIVISIE     CHAR(6)    NOT NULL
   PRIMARY KEY(TEAMNR)
   FOREIGN KEY(SPELERSNR)
   REFERENCES SPELERS
   ON UPDATE SET NULL
   ON DELETE SET NULL)
```

Als een speler verwijderd wordt in de SPELERS-tabel, wordt in de rij van de TEAMS-tabel waar dit nummer eveneens voorkomt, het spelersnummer vervangen door de NULL-waarde. En bij het verwijderen van een speler uit de SPELERS-tabel wordt het spelersnummer in de TEAMS-tabel door de NULL-waarde vervangen.

Belangrijke opmerking. In feite is de SET NULL-instructie incorrect. De reden is dat de SPELERSNR-kolom in de TEAMS-tabel gedefinieerd staat als NOT NULL. Er kunnen dus geen NULL-waarden ingevoerd worden. SQL zal dan ook de CREATE TABLE-instructie niet accepteren.

In de voorgaande voorbeelden gebruiken we steeds dezelfde acties voor UPDATE en DELETE. Dit is niet vereist. Er kan dus een refererende sleutel gedefinieerd worden met de refererende acties ON UPDATE RESTRICT en ON DELETE CASCADE.

4.4.2. Reservering van opslagruimte

Bij het creëren van een database moet niet aleen de databasestructuur worden beschreven, maar moet ook *fysieke opslagruimte* worden gereserveerd. Bij een database op PC hoeft alleen maar een directory te worden gecreëerd en een naam aan de database te worden gegeven ; opslagruimte wordt automatisch toegewezen door het besturingssysteem. Als het gaat om een multi-user database op een grotere computer moet er meestal meer werk worden gedaan. Er moet dan aangegeven worden welke tabellen op welke schijfeenheden worden opgeslagen. De fysieke toewijzing van tabellen aan opslagmedia is van belang voor de systeemprestaties.

4.4.3. Het vullen van de database met gegevens

Als de database gedefinieerd is en de opslagmedia zijn toegewezen, kan begonnen worden met het vullen van de database. Hoe dit in zijn werk gaat, hangt af van het soort DBMS dat wordt gebruikt. In het eenvoudigste geval staan de gegevens al in computerleesbare vorm en kunnen zij direct met behulp van faciliteiten van het DBMS in de database worden ingelezen. In het slechtste geval moeten alle gegevens met de hand worden ingebracht. Na of tijdens de invoer van de gegevens moeten deze worden gecontroleerd op juistheid. Dat is een arbeidsintensief en vervelend werkje, maar het is wel erg belangrijk. Bij grote databases is het vaak de moeite waard verificatieprogramma's te schrijven. Met die programma's worden aantallen records geteld, controletotalen berekend en andere correctheidscontroles uitgevoerd.

4.5. Relationele gegevensmanipulatie

Tot nu toe hebben we ons beziggehouden met het ontwerp van relationele databases en het implementeren van een dergelijk ontwerp naar een DBMS. Als we een applicatie werkelijk willen bouwen, hebben we een duidelijke ondubbelzinnige taal nodig voor het specifiëren van verwerkingsopdrachten. *Doel van een databasetoepassing is het raadplegen, creëren, wijzigen en verwijderen van objecten op aanwijzing van gebruikers, steeds rekening houdend met alle eisen ten aanzien van beveiliging en integriteit van de database*. In een database worden geen objecten opgeslagen. Wat wordt opgeslagen, zijn transformaties van

objecten in de vorm van verzamelingen van relatietabellen. Vooraleer een object kan worden bewerkt, moet het worden geconstrueerd uit deze onderliggende relaties. Men noemt dit *het materialiseren van objecten uit de database*.

De meeste DBMS-producten bieden niet de mogelijkheid informatie over objecten op te vragen ; wat wél wordt ondersteund door de meeste DBMS-producten is het opvragen van relaties. Verder zullen we kennismaken met SQL, een standaard vraagtaal. SQL is relatie-georiënteerd en gebruikers van een dergelijke taal moeten zelf zorgen voor het samenstellen van hun objecten uit de bij die objecten horende relaties.

4.5.1. Hulpmiddelen voor verwerkingsbesturing

We definiërden de primaire doelstelleing van een databasebasetoepassing als het raadplegen, creëren, wijzigen en verwijderen van objecten *op aanwijzing van gebruikers*. Het is de toepassing, de applicatie, die de gebruiker de mogelijkheid moet bieden de bewerkingen van de applicatie te sturen. Hiertoe staan twee hulpmiddelen ter beschikking.

Het ene hulpmiddel bestaat uit een verzameling van *commando's*, zoals :

```
UPDATE KLANT WHERE KLANTNUMMER = 12345
SET KLANTBEDRAG TO 1350
```

Voordeel van het gebruik van dergelijke commando's is dat deze meestal eenvoudig en direct zijn. Er zijn geen extra stappen nodig via menukeuzen, e.d. Nadeel is dat de eindgebruiker de commando's en de grammatica van een taal moet leren. Meestal wordt alleen voor een dergelijke commandogestuurde oplossing gekozen als veelvuldig gebruik wordt gemaakt van slechts een beperkt aantal commando's.

Een tweede hulpmiddel wordt gevormd door *keuzemenus*. De gebruiker kan steeds zijn keuze maken uit een aantal opties die in menuvorm worden aangeboden. Een voordeel van menus is dat de gebruiker geen commando's hoeft te onthouden en zelfs niet hoeft te weten welke commando's in een bepaalde situatie gebruikt zouden kunnen worden. Het systeem leidt de gebruiker 'aan de hand' door de keuzemogelijkheden die in een bepaalde situatie van toepassing zijn. Nadeel van menutoepassingen is dat ze op den duur omslachtig worden gevonden door meer ervaren gebruikers.

Nog niet zolang geleden is een nieuwe besturingsinterface ontwikkeld, die populair geworden is via Apple Macintosh-computers en Microsoft Windows. Het gaat om een zogeheten *grafische gebruikersinterface* (GUI), waarin op een standaardmanier gebruik wordt gemaakt van vensters, pictogrammen en muisbesturing. Omdat er intuïtief kan worden gewerkt en omdat alles steeds op dezelfde wijze is opgelost, zijn deze grafische interfaces gemakkelijk te gebruiken. Verwacht mag worden dat bij databasetoepassingen in de toekomst steeds meer van grafische interfaces gebruik zal worden gemaakt.

4.5.2. Beveiliging en integriteit

Onderdeel van de geformuleerde doelstelling was ook : *steeds rekening houdend met alle eisen ten aanzien van beveiliging en integriteit van de database*. Dit wil o.m. zeggen dat applicaties zo moeten worden ontworpen dat alleen bevoegde gebruikers toegelaten functies op toegelaten gegevens kunnen uitvoeren. Ook dit is gemakkelijker gezegd dan gedaan, maar het is wel iets waarmee de ontwerpers voortdurend rekening moeten houden.

Beveiliging en integriteit kunnen via applicaties op verschillende manieren worden bevorderd. Er kunnen *controles op randvoorwaarden* worden ingebouwd, en invoerschermen kunnen zo worden ontworpen dat de kans op fouten wordt verkleind. Als bijvoorbeeld in een veld maximaal vijfentwintig tekens zijn toegelaten, hoeft er op het invoerscherm ook maar ruimte voor de invoer van maximaal vijfentwintig tekens aanwezig te zijn. De menustructuur levert eveneens een bijdrage tot de beveiliging : alleen functies die in een bepaalde context zijn toegelaten hoeven te worden getoond.

4.5.3. Database-applicaties

Een database-applicatie is *een interface tussen een gebruiker en een database*, maar niet alle applicaties zijn gelijkwaardig. Men zou applicaties op een schaalverdeling kunnen plaatsen. Aan de ene kant staan de applicaties die volledig in dienst staan van de gebruiker. Deze applicaties verzamelen gegevens uit verschillende tabellen, combineren deze tot een objectstructuur en verzorgen wijzigingen via een uitgebreid GUI met een groot aantal functies. Tegelijkertijd controleren deze applicaties voortdurend de

integriteit van de database : ze zorgen ervoor dat geen randvoorwaarden worden overtreden en dat alleen toegelaten bewerkingen worden uitgevoerd. Dit alles is zo georganiseerd en gestructureerd dat de efficiëntie van de verwerking maximaal is.

Aan de andere zijde van de schaal staan de applicaties die niet veel meer doen dan de inhoud van één of meerdere tabellen tonen. In volgend deel maken we kennis met de taal SQL. Deze taal kan worden gebruikt als een soort minimale applicatie waarvan de enige functie het tonen van gegevens uit tabellen is. Mogelijkheden voor het weergeven van gegevens zoals hierboven beschreven en voor het controleren van de gegevensintegriteit ontbreken dan vrijwel volledig.

Database-applicaties kunnen op verschillende manieren worden ontworpen. Bij de ontwikkeling van een database-toepassing kunnen een drietal fasen worden onderscheiden : het bepalen van de functionele specificaties, het ontwerp en de implementatie. In de specificatiefase moeten de functies van de toepassing worden vastgesteld. Een hulpmiddel hierbij is het opstellen van gegevensstroomdiagrammen waarin gegevensstromen en gegevensverwerkende processen worden geïdentificeerd. Als de gewenste functies eenmaal zijn vastgesteld, moeten de applicaties worden ontworpen. De term 'applicatie-ontwerp' houdt in dit opzicht meer in dan het vaststellen van de structuur en de uit te voeren stappen in de programma's. Ook het ontwerp van gebruikersinterfaces, van formulieren, menustructuren en rapporten vallen hieronder. Nadat de applicaties ontworpen zijn, moeten ze worden geïmplementeerd. Dit kan op verschillende manieren gebeuren. Het ene uiterste is het schrijven van alle programma's in een derde-generatietaal zoals COBOL, en het andere uiterste is het volledig gebruik maken van ingebouwde DBMS-fucnties. Veel applicaties zullen ergens tussen deze twee uitersten in liggen : zij combineren zelfgeschreven programma's met scherm- en rapportgeneratoren van het DBMS.

4.5.3.1. Object- en viewmaterialisatie

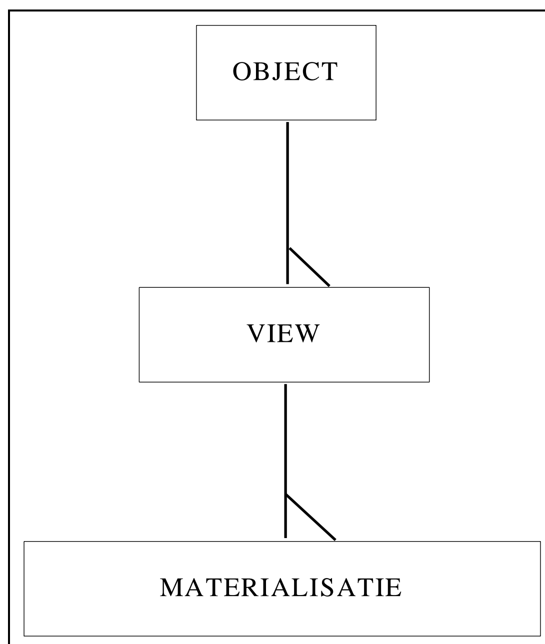
Een van de functies van een database-applicatie is het materialiseren van objecten. Objecten worden ge(re)construeerd uit de gegevens in de database. Dit proces wordt *objectmaterialisatie* genoemd. Het is binnen applicaties een belangrijke functie, omdat gebruikers niet hoeven te weten in welke tabellen bepaalde gegevens zijn opgeslagen en hoe deze gegevens gecombineerd moeten worden.

Gebruikers willen zelden alle gegevens van een object in één keer op het scherm zien ; meestal zijn ze geïnteresseerd in een deelverzameling. Verschillende gebruikers werken anderzijds wel met dezelfde objecten, maar zien die objecten vanuit een verschillend perspectief.

De deelverzameling van gegevens over een object die gebruikers voor een bepaald doeleinde willen zien, noemt men een *objectview*, een *gebruikersview*, of kortweg een *view*. Een view wordt beschreven door middel van het object waarop de view betrekking heeft en door een opsomming van de attributen die vanuit de view zichtbaar zijn.

Een view is een lijstje van attributen, maar meestal willen gebruikers meer dan een lijstje : zij willen de gegevens in een bepaalde opmaak op het scherm zien. De combinatie van een view met opmaakgegevens noemt men een *materialisatie*. Een materialisatie van een view bestaat uit de view zelf in combinatie met de definitie van de formulieropmaak (kleuren, kaders, teksten, enz.). We kunnen dus stellen dat het de taak van applicatieprogramma's is objectviews te materialiseren.

Objecten, views en materialisaties hebben een hiërarchische samenhang zoals hieronder schematisch voorgesteld. *Er kunnen meer views op een object bestaan en bij iedere view kunnen meerdere materialisaties gedefinieerd zijn*.



4.5.3.2. Formulieren

Met een formulier bedoelen we in een bepaalde opmaak op een beeldscherm geprojecteerde gegevens. Formulieren worden gebruikt voor het invoeren, wijzigen en raadplegen van gegevens. Of een formulier gemakkelijk te gebruiken is, of overzichtelijk en foutgevoelig is, hangt af van de kwaliteit van het ontwerp. In deze paragraaf geven we een aantal richtlijnen voor het ontwerpen van goede formulieren.

Een formulier lijkt natuurlijker en is gemakkelijker te gebruiken als de formulierstructuur overeenkomt met de structuur van het object. Bij het ontwerpen van een formulier kan men in het algemeen alle gegevens uit eenzelfde relatie het beste in hetzelfde gebied van het formulier plaatsen. Als de database in domein/sleutel-normaalvorm staat, heeft iedere tabel betrekking op een bepaald onderwerp en dan staan dus ook de gegevens over dit onderwerp bij elkaar.

Formulieren moet bij voorkeur zo zijn opgemaakt dat zij automatisch aanleiding geven tot juiste handelingen, terwijl onjuiste handelingen moeilijk of zelfs onmogelijk worden gemaakt (in Microsoft Access is het bijvoorbeeld mogelijk om validatieregels voor velden te definiëren).

In een GUI-omgeving kunnen een aantal specifieke mogelijkheden het gebruik van databasetoepassingen aanzienlijk vergemakkelijken. Een keuzelijstje in een GUI-omgeving toont een aantal elementen waaruit de gebruiker kan kiezen door erop te klikken. Sommige lijstjes tonen een vast aantal elementen, andere kunen door de gebruiker worden aangevuld. Dergelijke keuzelijstjes hebben een aantal voordelen boven invoervensters. Mensen vinden het over het algemeen gemakkelijker te herkennen en te kiezen, dan te onthouden. Verder kan de keuze beperkt worden waardoor de gebruiker geen onbekende gegevens kan invoeren.

Een keuzerondje of "radio button" is een hulpmiddel voor het kiezen van een mogelijkheid uit een aantal elkaar uitsluitende alternatieven. De gemaakte keuze moet door het applicatieprogramma worden verwerkt en in de database worden opgeslagen.

Een aankruisvakje of "check box" lijkt op een keuzerondje, maar nu sluiten de verschillende alternatieven elkaar niet uit en mag er dus meer dan één keuze worden gedaan.
Bij het ontwerpen van formulieren moet men ook letten op de wijze waarop de cursor zich gedraagt. De cursor moet zich op een 'natuurlijke' manier over het scherm bewegen. Dit betekent dat de cursor de velden moet aandoen in de volgorde waarin de gebruiker de gegevens invoert of leest. Als zich een specifieke situatie voordoet, als bijvoorbeeld een fout wordt geconstateerd,

moet de cursor zich eveneens op een logische manier gedragen en bijvoorbeeld een toegelaten waarde aanwijzen in een lijstje van mogelijke waarden. Vervolgens moet de cursor terugkeren naar de oorspronkelijke positie. De acties die gekoppeld zijn aan speciale toetsen zoals de Escape-toets en de functietoetsen moeten consistent en in alle situaties hetzelfde zijn. Als de Escape-toets wordt gebruikt om formulieren te verlaten, moet dit altijd op deze manier gebeuren en niet ergens anders met bijvoorbeeld Alt-X. Als dit niet consequent gebeurt, werkt dit verwarrend voor de gebruiker en kunnen gemakkelijker fouten optreden. Deze opmerkingen lijken vanzelfsprekend, maar toch zijn het juist deze details die maken dat men gemakkelijk met een systeem leert werken.

4.5.3.3. Rapporten

De richtlijnen voor het ontwerpen van goede rapporten komen overeen met die voor het ontwerp van goede formulieren. Een rapport kan in feite worden gezien als een niet wijzigbaar raadpleegformulier. Net als formulieren dienen rapporten de structuur van de eraan ten grondslag liggende objecten te weerspiegelen. Dit betekent dat ook hier gegevens uit eenzelfde tabel in het rapport bij elkaar moeten worden teruggevonden.

4.5.3.4. Relationele datamanipulatietalen

Eerder werd er reeds op gewezen dat wanneer men een applicatie werkelijk wil bouwen, een taal nodig is voor het specificeren van verwerkingsopdrachten. Er zijn op dit moment vier verschillende methoden voor het manipuleren van relationele gegevens. De eerste methode is de *relationele algebra*, waarin operatoren worden gedefinieerd voor het bewerken van relaties (deze operatoren zijn te vergelijken met +, -, * en / zoals we ze kennen in de 'gewone' algebra). Relationele algebra is niet eenvoudig in het gebruik, o.m. omdat het een procedurele taal betreft. Dit betekent dat we niet alleen moeten specifiëren *wat* we zoeken, maar ook *hoe* we willen zoeken. In commerciële omgevingen wordt relationele algebra, omdat ze zo ingewikkeld is, zelden toegepast, hoewel er DBMS-producten zijn die de mogelijkheid bieden ermee te werken. Toch is enig inzicht in relationele algebra belangrijk, met name omdat ook een taal als SQL, die wel veel wordt gebruikt, erop is gebaseerd.

De *relatierekening* ("relational calculus") is een tweede middel voor relationele gegevensmanipulatie. Relationele calculus is niet-procedureel. Dit betekent dat men alleen aangeeft welk resultaat men zoekt, zonder aan te geven op welke wijze dit moet worden gevonden. Bij commerciële databasetoepassingen wordt vrijwel geen gebruik gemaakt van de relatierekening. Niettemin is de relatierekening aantrekkelijk omdat ze niet-procedureel is ; men hoeft alleen het doel en niet de methode te specifiëren. Men zocht daarom naar andere non-procedurele technieken en dit leidde tot het derde en vierde type DML (*Data Manipulation Language*).

Transformatiegeoriënteerde talen vormen een klasse non-procedurele talen die invoer in de vorm van relaties transformeren naar resultaten in de vorm van één enkele relatie. Deze talen bieden de mogelijkheid om op een vrij eenvoudige manier te formuleren wat men zoekt in termen van de invoergegevens. De talen SQUARE, SEQUEL en het van deze laatste afgeleide taal SQL zijn alle voorbeelden van transformatiegeoriënteerde talen. In deel 2 zullen we SQL nader onder de loep nemen.

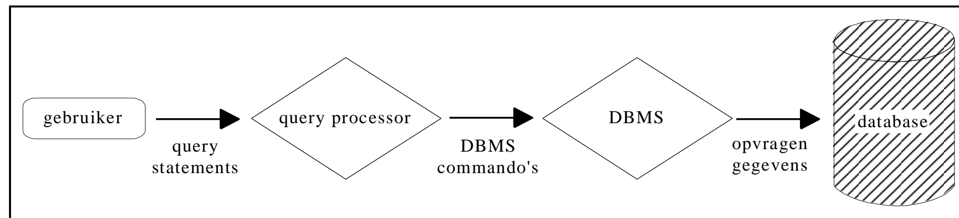
De vierde categorie DML's wordt gevormd door *grafisch georiënteerde systemen*. Voorbeelden zijn Paradox van Borland en Microsoft Access. Bij deze systemen krijgt men een grafische voorstelling van een relatie te zien. De gebruiker vult een voorbeeld in van de gegevens die hij of zij wil zien en het systeem levert alle gegevens die voldoen aan dit voorbeeld. Men noemt deze werkwijze *QBE* of "query by example".

4.5.3.5. Interfaces met het DBMS

Eerder beschreven we een aantal manieren waarop een gebruiker met een database kan communiceren. Men kan in de eerste plaats gebruik maken van de formulier- en rapportagemogelijkheden van het DBMS. In de tweede plaats kan men een database benaderen via een vraagtaal (query-taal) en ten slotte kan men gebruik maken van applicatieprogramma's die het DBMS benaderen via commando's.

De meeste relationele DBMS-producten bevatten gereedschappen voor het definiëren van *formulieren* (op het beeldscherm weergegeven gegevensverzamelingen). Sommige formulieren worden automatisch bij het definiëren van een tabel gecreëerd, andere moeten door de bouwer van de applicatie worden gemaakt met behulp van schermdefinitiegereedschappen zoals de Wizards in Microsoft Access. De opmaak kan in tabelvorm zijn, zoals bij een spreadsheet, met meer rijen tegelijkertijd op het scherm. Een scherm kan ook de gegevens van één rij tonen.

Een tweede interface met een database wordt gevormd door query/update-talen (vraag/wijzigtalen). Meestal spreekt men, ook indien wijzigen van gegevens tot de mogelijkheden behoort, van *query-talen* of *vraagtalen*.



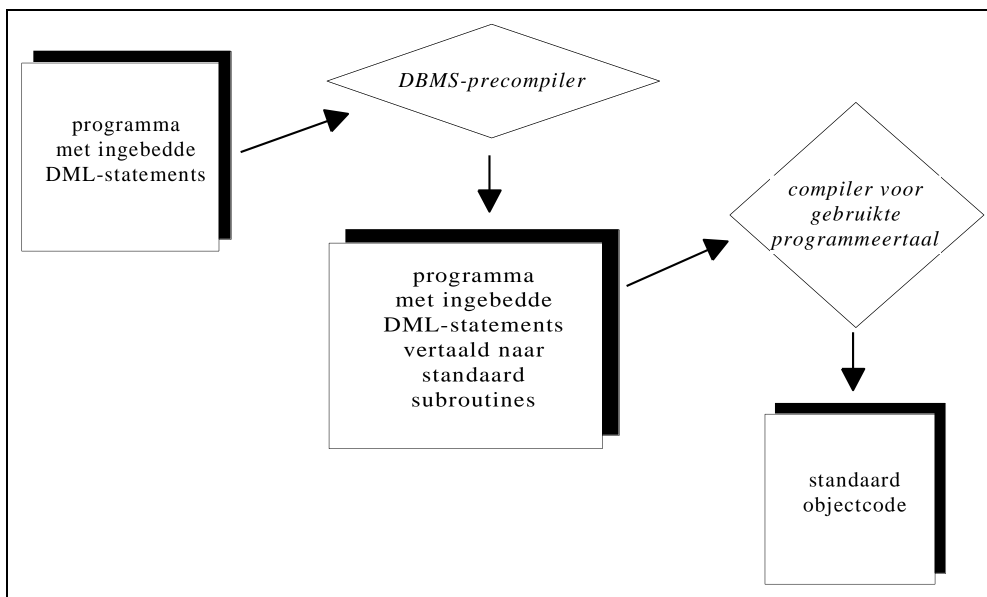
Bij gebruik van een vraagtaal worden acties op de gegevens in de database met behulp van commando's gespecifieerd. Het DBMS vertaalt deze commando's en voert de bijbehorende acties uit. De taal SQL is de meest gebruikte query-taal. Query-talen zijn ontstaan omdat er een behoefte was aan een eenvoudige en algemeen bruikbare interface met de database, waarbij niet geprogrammeerd hoeft te worden in een traditionele procedurele taal.

De derde manier van communiceren met een database is via applicatieprogramma's geschreven in een taal als COBOL, Pascal of C. Ook worden veel applicatieprogramma's geschreven in speciale bij het DBMS geleverde programmeertalen, zoals bijvoorbeeld de dBASE-taal die voor veel PC-toepassingen wordt gebruikt.

Bij communiceren met het DBMS door middel van een applicatieprogramma zijn twee methoden te onderscheiden. De eerste methode maakt gebruik van subroutine-aanroepen vanuit het applicatieprogramma naar procedures in een bij het DBMS geleverde subroutinebibliotheek. Om bijvoorbeeld bepaalde gegevens uit een tabel te lezen, roept het programma een DBMS read-routine aan met daarbij de benodigde parameters, zoals de naam van de tabel, de te lezen velden, eventuele selectiecriteria e.d. In de praktijk blijkt het benaderen van een database via subroutines nogal gecompliceerd te zijn. Het ontwikkelen van dergelijke applicatieprogramma's kost veel tijd en geld.

Daarom wordt vaak een tweede methode gebruikt om programma's met de database te laten communiceren : er wordt een verzameling *commando's* bij het DBMS geleverd die kunnen opgenomen worden in de applicatieprogrammatuur. Deze commando's zijn speciaal ontworpen voor het werken met databases en vormen zelf geen onderdeel van een standaard programmeertaal. Het applicatieprogramma waarin de databasecommando's zijn opgenomen, wordt vervolgens aangeboden aan een meegeleverde precompiler (voorvertaler). Deze precompiler vertaalt de commando's in programmacode in de voor de applicatie gebruikte

programmeertaal en in door deze taal te verwerken subroutine-aanroepen. De precompiler zorgt eveneens voor de bijbehorende parameterwaarden en voor het reserveren van geheugenruimte die door het programma en het DBMS gemeenschappelijk wordt gebruikt. Na verwerking door de precompiler kan het programma gecompileerd worden door de standaardcompiler van de gebruikte programmeertaal. Hieronder wordt dit proces schematisch voorgesteld.



Ook SQL wordt, behalve voor het direct verwerken van queries, gebruikt als taal voor het benaderen van gegevens vanuit een applicatieprogramma. SQL-statements worden dan ingebed in het programma ("imbedded SQL") en zij worden door een precompiler naar subroutines vertaald. Dit heeft tot voordeel dat men maar één nieuwe taal hoeft te leren.

Nadeel van het gebruik van SQL binnen een applicatie is dat SQL een transformatiegeoriënteerde taal is, die relaties als input accepteert, deze bewerkt en vervolgens een relatie als resultaat oplevert. De verwerkingseenheid van SQL is dus steeds een gehele relatie (tabel), terwijl een applicatieprogramma meestal met rijen (records) werkt (d.w.z. de gegevens van een record inleest, deze bewerkt en vervolgens de gegevens van een volgende record inleest, enz.). De oriëntatie van SQL sluit dus niet aan op die van de meeste voor applicaties gebruikte programmeertalen. Het verschil in oriëntatie tussen SQL (relatie-georiënteerd) en programmeertalen (rij- of recordgeoriënteerd) wordt vaak opgelost door binnen het applicatieprogramma het resultaat van een SQL-opdracht te zien als een

bestand dat door het programma verder bewerkt kan worden. Veronderstel dat onderstaande SQL-opdracht in een applicatieprogramma wordt ingebed :

```
SELECT   NAAM, LEEFTIJD
FROM     PATIENT
WHERE    LEEFTIJD >= 50
```

Het resultaat is een tabel met twee kolommen en N rijen. In het programma wordt verondersteld dat een bestand met N records met elk twee velden is gegenereerd. Dit bestand kan vervolgens door het programma rij voor rij verwerkt worden alsof het om een sequentiële file gaat.

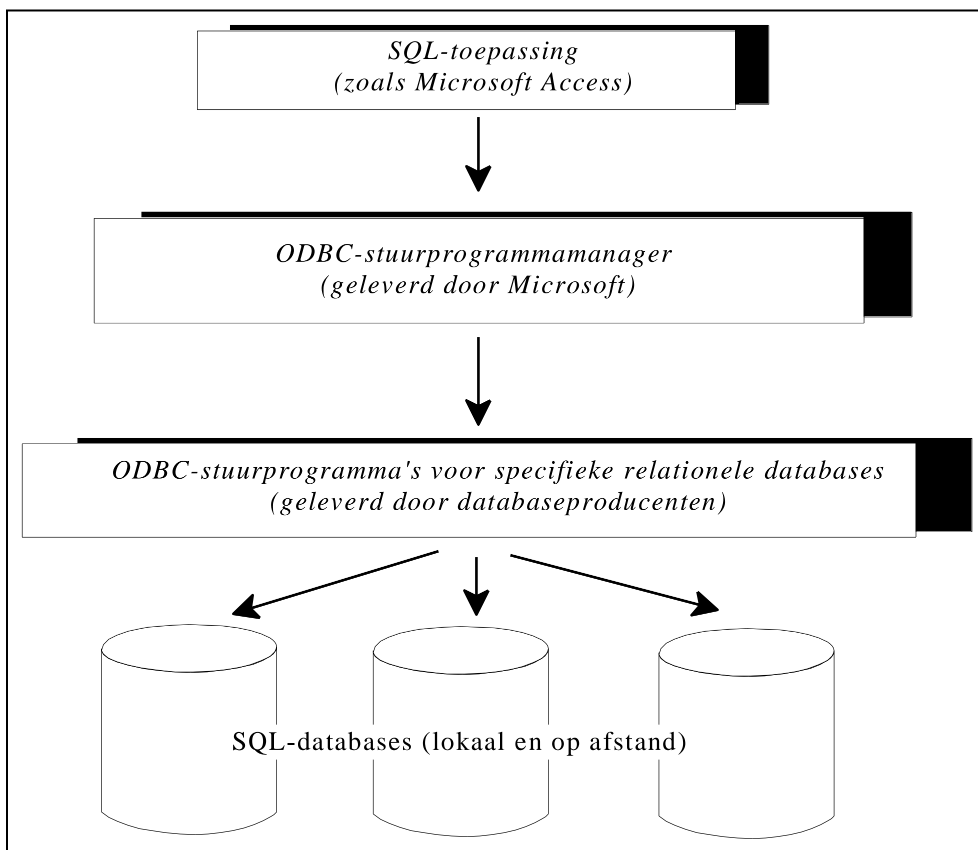
4.5.3.6. Open Database Connectivity (ODBC)

Hoewel PC-producten gebruikt worden als zelfstandige systemen voor het bewerken van databases, is een van de sterke punten van deze producten dat deze niet alleen toelaten met lokaal (d.w.z. op de harde schijf van de PC) opgeslagen databases te werken, maar eveneens met gegevens die bijvoorbeeld op een server worden bijgehouden.

Kijkt men onder de motorkap van bijvoorbeeld Microsoft Access, dan ziet men dat het programma SQL gebruikt voor het lezen, toevoegen, bijwerken en verwijderen van gegevens. In ideale omstandigheden zou elk product dat SQL 'spreekt' kunnen 'praten' met alle andere producten die SQL verstaan. In dit geval zou men een toepassing kunnen maken door gebruikmaking van één databasetaal waarbij men werkt met gegevens van uiteenlopende RDBMS-systemen. Er bestaan standaarden voor SQL, maar de meeste softwarebedrijven gebruiken varianten van of uitbreidingen op de taal voor het afhandelen van de speciale mogelijkheden van hun producten. Bovendien waren verscheidene producten al ontwikkeld voordat de standaarden goed en wel waren vastgelegd, dus de bedrijven die deze producten maken, hebben hun eigen syntaxis voor SQL bedacht, die afwijkt van de standaard. Het kan zijn dat een SQL-opdracht voor Microsoft SQL Server moet worden aangepast voordat de opdracht kan worden uitgevoerd door DB2 en Oracle.

Om dit probleem op te lossen hebben diverse invloedrijke hard- en softwarebedrijven - meer dan dertig, waaronder Microsoft Corporation - zich enkele jaren geleden verenigd in de *SQL Access Group*. Het doel van deze groep was het definiëren van een algemene standaardimplementatie van SQL die alle producten van de leden van de groep zouden kunnen gebruiken om

met elkaar te 'praten'. De bedrijven ontwikkelden de *Common Language Interface* (CLI) voor alle hoofdvarianten van SQL en verplichtten zich ertoe ondersteuning voor CLI in te bouwen in hun producten. Begin 1992 demonstreerden meer dan een dozijn van deze bedrijven gezamenlijk deze nieuwe mogelijkheid. Microsoft formaliseerde CLI voor de producten van het besturingssysteem Microsoft Windows. Microsoft noemt deze geformaliseerde interface *Open Database Connectivity* (ODBC). Bij Access levert Microsoft het standaard ODBC-stuurprogramma. Microsoft heeft ook samengewerkt met andere databaseproducenten voor het ontwikkelen van stuurprogramma's voor andere databases. In het schema op blz. 74 wordt de architectuur van Microsoft ODBC voorgesteld.



5. Structured query language

Structured Query Language (gestructureerde vraagtaal) of SQL is tegenwoordig de belangrijkste *relationele gegevensmanipulatietaal*. Het ANSI (American National Standards Institute) heeft deze taal als standaard gekozen voor het manipuleren van databases. DBMS-producten zoals DB2, Oracle, Sybase, SQL Server, Paradox en Microsoft Access gebruiken alle een vorm van SQL. Ook voor het uitwisselen van gegevens tussen computersystemen wordt meer en meer van SQL gebruik gemaakt. Voor de meeste computersystemen bestaat wel een versie van SQL en dus kunnen verschillende systemen via SQL onderling berichten en antwoorden versturen. Het is waarschijnlijk dat het gebruik van SQL voor dit soort gegevensuitwisseling in de nabije toekomst nog verder zal toenemen.

De ontwikkeling van SQL is begonnen in het midden van de jaren zeventig op het IBM-onderzoeksinstituut in San Jose. Deze eerste versie heette SEQUEL. Na diverse versies van SEQUEL werd de taal in 1980 SQL gedoopt. Sindsdien hebben tal van andere leveranciers SQL-producten ontwikkeld. Ook het ANSI blijft SQL volgen en regelmatig verschijnt een aangepaste versie van de SQL-standaard. Vanuit een marketingstandpunt wordt de ANSI-standaard zonder extra's als onvoldoende aanzien. Vandaar dat verschillende DBMS-producten op punten van de ANSI-standaard afwijken omdat concurrerende producten extra mogelijkheden toevoegen.

SQL-commando's kunnen interactief worden gebruikt voor query's, of zij kunnen worden ingebed in applicatieprogramma's. In dit laatste geval worden zij vertaald door een precompiler. SQL is immers geen programmeertaal : het is een *gegevensbenaderingstaal*.

5.1. Componenten van de SELECT-instructie

De belangrijkste en meest gebruikte SQL-instructie is *SELECT*. Met deze instructie worden gegevens in tabellen geraadpleegd. Een SELECT-instructie kan samengesteld worden uit een aantal (verplichte en niet-verplichte) componenten :

SELECT-component
FROM-component
[WHERE-component]
[GROUP BY-component]
[HAVING-component]
[ORDER BY-component]

De volgende regels zijn van belang bij het formuleren van SELECT-instructies.

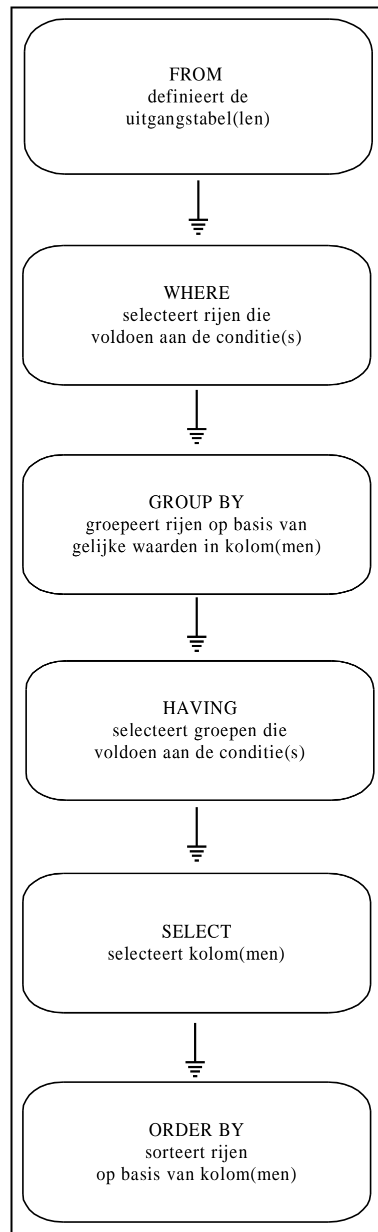
1. Elke SELECT-instructie bestaat minimaal uit twee componenten : de SELECT- en de FROM-component. De andere componenten zijn niet verplicht.
2. De volgorde van de componenten staat vast ; een GROUP BY-component mag nooit vóór een FROM- of WHERE-component staan en de ORDER BY-component is (bij gebruik) altijd de laatste.

3. Een HAVING-component mag alleen gebruikt worden als een GROUP BY-component gebruikt wordt.

SQL zal de SELECT-instructies niet verwerken in de volgorde zoals hierboven beschreven, maar zal trachten zoveel mogelijk componenten tegelijkertijd uit te voeren om de verwerking te versnellen. Hoe instructies verwerkt worden, wordt samengevat in het schema op volgende pagina.

SELECT specifieert kolommen, in relationele termen de attributen, FROM specifieert één of meerdere tabellen, in relationele termen, de relatie(s), en WHERE specifieert de conditie(s) waaraan de rijen moeten voldoen. Benadrukt moet worden dat het resultaat van een vraag weer een tabel is.

De bewerking die door middel van de voorwaarde(n) in de WHERE-regel uitgevoerd wordt, wordt een selectie genoemd. Het is misschien verwarrend, maar selectie vindt niet plaats in de SELECT-regel, maar in de WHERE-regel. De term 'selectie' heeft dus betrekking op de rijen in tabellen, de term 'projectie' op de kolommen (kolommen worden geselecteerd door de SELECT-instructie). Met de WHERE-component worden rijen geselecteerd. Het tussenresultaat van deze component vormt een horizontale deelverzameling van een tabel. De SELECT-component daarentegen selecteert kolommen en het resultaat vormt een verticale deelverzameling van een tabel.



5.2. Relaties tussen tabellen

In hetgeen volgt gaan we dieper in op SQL-opdrachten die betrekking hebben op meerdere tabellen. De zgn. "join" is, in de meest algemene zin, een samenvoeging van tabellen. Bij een join van tabellen worden de rijen uit de betrokken tabellen (horizontaal) aan elkaar geplakt. Het resultaat is weer een tabel. Een SELECT-instructie kan een *join* genoemd worden als ten eerste de

FROM-component minimaal twee tabelspecificaties bevat en ten tweede de WHERE-component minstens één conditie bevat waarin kolommen in de verschillende tabellen met elkaar vergeleken worden.

Indien we zonder vermelding van voorwaarden een join maken van bijvoorbeeld twee tabellen, dan wordt in feite het product gemaakt van die twee tabellen : indien in beide tabellen 10 waarnemingen opgenomen zijn, is het resultaat van de join een nieuwe tabel met 100 waarnemingen *(cartesisch product)*. Meestal is zo'n join zinloos (behalve wanneer een tabel met zichzelf wordt vergeleken, cfr. infra). Daarom wordt aan de join een voorwaarde toegevoegd, een zgn. joinvoorwaarde.

De kolommen die in een SELECT-instructie voor de join zorgen, worden de *joinkolommen* genoemd. Tussen joinkolommen bestaat altijd een soort relatie. Als K1 en K2 twee kolommen zijn, dan zijn tussen K1 en K2 vier soorten relaties mogelijk.

1. De populaties van K1 en K2 zijn *gelijk*.

2. De populatie van K1 is een *deelverzameling* van de populatie van K2, of omgekeerd.

3. De populaties van K1 en K2 zijn *conjunct*, d.w.z. ze hebben enkele gemeenschappelijke waarden.

4. De populaties van K1 en K2 zijn *disjunct*, d.w.z. ze hebben geen enkele waarde gemeenschappelijk.

De aard van de relatie tussen de joinkolommen is bepalend voor het resultaat van de SELECT-instructie (cfr. infra).

5.2.1. Soorten joins

Om records uit verschillende tabellen met elkaar te verbinden, worden speciale velden gebruikt : sleutelvelden. Er zijn twee belangrijke soorten sleutels : *primaire sleutels* en *vreemde of verwijzende sleutels*. De relatie wordt steeds gelegd tussen de primaire sleutel in de ene tabel en de vreemde sleutel in de andere tabel.

Een join met een voorwaarde waarin tabellen gekoppeld worden met behulp van de gelijkheidsoperator (=), wordt een *equijoin* genoemd. Tabellen kunnen ook gekoppeld worden op basis van andere vergelijkingsoperatoren dan de gelijkheidsoperator. In dit geval spreekt men van *non-equijoins* of *thèta-joins*.

Inner joins, een vorm van equijoin, tonen alleen die records van een tabel die overeenkomstige records in een andere tabel bevatten. De overeenkomst tussen records wordt bepaald door identieke waarden in de velden. Een andere type equijoin, is de *outer join*. Outer joins tonen records aan één kant van de verbinding, ongeacht het overeenkomstige record aan de andere kant van de verbinding bestaat. In dit opzicht wordt een onderscheid gemaakt tussen een "left outer join" en een "right outer join". Indien alle records aan beide kanten van de verbinding in het resultaat opgenomen worden, spreekt men van een "full outer join".

Self-joins of *auto-joins* leggen een relatie binnen een enkele tabel. Hiertoe wordt in de FROM-component een pseudoniem voor de tabel gebruikt waardoor elke rij uit de tabel kan vergeleken worden met elke rij uit de(zelfde) tabel.

5.2.2. Relaties tussen joinkolommen

We zullen voor elk soort relatie tussen joinkolommen aangeven wat de invloed is van de soort relatie op het resultaat van een joinbewerking. We illustreren de verschillende SQL-opdrachten aan de hand van twee tabellen, nl. de tabel SPELERS en de tabel BIJDRAGE. Beide tabellen worden met elkaar verbonden via het veld SPELERSNR.

5.2.2.1. De populaties van de joinkolommen zijn hetzelfde

Veronderstel dat de tabel SPELERS en de tabel BIJDRAGE volgende inhoud hebben :

SPELERS.SPELERSNR	SPELERS.PLAATS
6	Den Haag
44	Rijswijk
104	Zoetermeer

BIJDRAGE.SPELERSNR	BIJDRAGE.BEDRAG
6	100
44	75
44	25
44	30
104	50

De SQL-opdracht :

```
SELECT     SPELERS.SPELERSNR, BIJDRAGE.BEDRAG
FROM       SPELERS, BIJDRAGE
WHERE      SPELERS.SPELERSNR = BIJDRAGE.SPELERSNR
```

heeft als resultaat :

SPELERSNR	BEDRAG
6	100
44	75
44	25
44	30
104	50

Een join van dit type wordt een *inner (equi)join* genoemd omdat elke rij uit de tabel SPELERS verbonden wordt met elke rij uit de tabel BIJDRAGE op voorwaarde dat een verbinding kan gelegd worden op basis van het veld SPELERSNR. Indien de populaties van de joinkolommen hetzelfde zijn, levert een *outer join* hetzelfde resultaat op als een inner join. Reden hiervoor is dat geen van beide tabellen een rij bevat met een spelersnummer dat niet in de andere tabel voorkomt.

5.2.2.2. De populatie van een joinkolom is een deelverzameling

Veronderstel dat de twee tabellen onderstaande inhoud hebben (waarbij de populatie van SPELERSNR in de tabel BIJDRAGE een deelverzameling is van de populatie van SPELERSNR in de de tabel SPELERS) :

SPELERS.SPELERSNR	SPELERS.PLAATS
6	Den Haag
44	Rijswijk
104	Zoetermeer

BIJDRAGE.SPELERSNR	BIJDRAGE.BEDRAG
6	100
104	50

De SQL-opdracht

```
SELECT     SPELERS.SPELERSNR, BIJDRAGE.BEDRAG
FROM       SPELERS, BIJDRAGE
WHERE      SPELERS.SPELERSNR = BIJDRAGE.SPELERSNR
```

heeft als resultaat :

SPELERSNR	BEDRAG
6	100
104	50

Alleen spelers die in beide tabellen en dus in de doorsnede van de twee tabellen voorkomen, worden in het resultaat van de *inner join*-opdracht opgenomen. Nummer 44 behoort niet tot deze doorsnede en komt dus niet in het resultaat voor. Het uitvoeren van een outer join heeft in dit geval wél een verschillende resultaat tot gevolg.

Om nummer 44 toch in het resultaat op te nemen, dient een *outer join*-opdracht uitgevoerd te worden .

```
SELECT     SPELERS.SPELERSNR, BIJDRAGE.BEDRAG
FROM       SPELERS, BIJDRAGE
WHERE      SPELERS.SPELERS.NR = BIJDRAGE.SPELERSNR
UNION
SELECT     SPELERS.SPELERSNR, 0
FROM       SPELERS
WHERE      SPELERSNR NOT IN
           (SELECT SPELERSNR FROM BIJDRAGE)
```

Bovenvermelde opdracht heeft als resultaat :

SPELERSNR	BEDRAG
6	100
104	50
44	0

Een join-opdracht zoals hierboven wordt ook een *left outer join* genoemd omdat alle records uit de linker tabel, d.w.z. de tabel die het eerst in de FROM-component vermeld wordt, in het resultaat moeten worden opgenomen. Worden alle records van de tweede, rechter tabel opgenomen dan wordt van een *right outer join* gesproken en als alle records van de twee tabellen in het resultaat moeten voorkomen wordt van een *full outer join* gesproken.

In bovenstaand voorbeeld is een right outer join overbodig omdat alle waarden in het veld SPELERSNR eveneens voorkomen in het veld SPELERSNR van de tabel SPELERS.

5.2.2.3. De populaties van de joinkolommen zijn conjunct

Veronderstel dat de tabel SPELERS en de tabel BIJDRAGE onderstaande inhoud hebben :

SPELERS.SPELERSNR	SPELERS.PLAATS
6	Den Haag
44	Rijswijk
104	Zoetermeer

BIJDRAGE.SPELERSNR	BIJDRAGE.BEDRAG
6	100
104	50
8	25

Met 'conjuct' wordt bedoeld dat de populaties van de beide joinkolommen gedeeltelijk dezelfde zijn : sommige waarden komen in beide tabellen voor en sommige waarden komen enkel in de ene en niet in de andere tabel voor, en omgekeerd. Indien een inner join wordt uitgevoerd zullen enkel de records met SPELERSNR 6 en 104 in het resultaat opgenomen worden. Beide tabellen hebben enkel deze waarden voor het veld SPELERSNR gemeenschappelijk.

Veronderstel dat we een *right outer join* wensen te voeren (alle records uit de tabel BIJDRAGE moeten in het resultaat opgenomen worden) waarin SPELERS.PLAATS, BIJDRAGE.SPELERSNR en BIJDRAGE.BEDRAG opgenomen zijn, dan wordt onderstaande SQL-opdracht gespecifieerd :

```
SELECT     SPELERS.PLAATS, BIJDRAGE.SPELERSNR,
           BIJDRAGE.BEDRAG
FROM       SPELERS, BIJDRAGE
WHERE      SPELERS.SPELERSNR = BIJDRAGE.SPELERSNR
UNION
SELECT     'niet gekend', BIJDRAGE.SPELERSNR,
           BIJDRAGE.BEDRAG
FROM       BIJDRAGE
WHERE      BIJDRAGE.SPELERSNR NOT IN
           (SELECT SPELERSNR FROM SPELERS)
```

Om een *full outer join* uit te voeren waarin voormelde velden in het resultaat voorkomen, wordt onderstaande SQL-opdracht gespecifieerd :

```
SELECT     SPELERS.PLAATS, BIJDRAGE.SPELERSNR,
           BIJDRAGE.BEDRAG
FROM       SPELERS, BIJDRAGE
WHERE      SPELERS.SPELERSNR = BIJDRAGE.SPELERSNR
UNION
SELECT     SPELERS.PLAATS,0,0
FROM       SPELERS
WHERE      SPELERSNR NOT IN
           (SELECT SPELERSNR FROM BIJDRAGE)
UNION
SELECT     'niet gekend', BIJDRAGE.SPELERSNR,
           BIJDRAGE.BEDRAG
FROM       BIJDRAGE
WHERE      BIJDRAGE.SPELERSNR NOT IN
           (SELECT SPELERSNR FROM SPELERS)
```

5.2.2.4. De populaties van de joinkolommen zijn disjunct

Een vierde mogelijkheid tenslotte is dat populaties van de joinkolommen helemaal geen overeenkomst vertonen : de records van de tabellen die in de join betrokken worden hebben voor de joinkolom(men) geen gemeenschappelijke waarden. In dit geval heeft een join weinig zin. Er kan dan volstaan worden met een UNION tussen beide tabellen.

Veronderstel dat de tabel SPELERS en de tabel BIJDRAGE volgende inhoud hebben :

SPELERS.SPELERSNR	SPELERS.PLAATS
6	Den Haag
44	Rijswijk
104	Zoetermeer

BIJDRAGE.SPELERSNR	BIJDRAGE.BEDRAG
29	100
186	50
220	25

Als we de tabel SPELERS en de tabel BIJDRAGE wensen samen te voegen, d.w.z. onder elkaar plakken, dan wordt onderstaande SQL-opdracht uitgevoerd :

```
SELECT     SPELERS.SPELERSNR, SPELERS.PLAATS, 0
FROM       SPELERS
UNION
SELECT     BIJDRAGE.SPELERSNR, 'niet gekend',
           BIJDRAGE.BEDRAG
FROM       BIJDRAGE
```

5.2.3. Uitbreiding

In hetgeen volgt zullen we enkele van de hierboven behandelde SQL-opdrachten verder illustreren aan de hand van de inhoud van twee tabellen, nl. de tabel WERKN en de tabel AFDELING. Beide tabellen kunnen gekoppeld worden door de waarden in het veld AFDCODE. Voor de uitwerking van de hierna te vermelden SQL-opdrachten is gebruik gemaakt van de versie 2.6 van Microsoft Foxpro for Windows (omdat elk onderdeel van een SQL-opdracht in Microsoft Foxpro for Windows 2.6 afgesloten wordt met een puntkomma, zullen we deze notatie eveneens opnemen in de SQL-commando's).

5.2.3.1. Inner join

```
SELECT     WERKN.NAAM, AFDELING.AFDCODE ;
FROM       WERKN, AFDELING ;
WHERE      WERKN.AFDCODE = AFDELING.AFDCODE ;
INTO       TABLE INNER.DBF
```

5.2.3.2. Autojoin

Om de naam, de afdelingscode en het aantal jaren in dienst te rapporteren van de werknemers die minder lang in dienst zijn dan werknemer WASIMAN, wordt gebruik gemaakt van een autojoin.

```
SELECT     W.NAAM, W.AFDCODE,W.IN_DIENST ;
FROM       WERKN W, WERKN N ;
WHERE      N.NAAM = "WASIMAN" ;
AND        W.IN_DIENST < N.IN_DIENST ;
INTO       TABLE AUTOJOIN.DBF
```

5.2.3.3. Tellen van dubbels

Voor het tellen van dubbel toegekende identificatienummers (veld IDNR) in tabel WERKN wordt gegroepeeerd op de waarden van het veld IDNR. Voor elk van die waarden wordt het aantal keer dat een waarde gebruikt werd, gerapporteerd m.b.v. de COUNT-functie.

```
SELECT     IDNR, COUNT(IDNR) ;
FROM       WERKN ;
GROUP BY   IDNR ;
INTO       TABLE DUBBELS.DBF
```

In bovenstaande tabel wordt het aantal keer dat een identificatienummer gebruikt wordt, weergegeven in de kolom CNT. Voor elk identificatienummer uit de tabel WERKN wordt vermeld hoeveel keer het betreffende nummer in de tabel voorkomt. Om enkel de identificatienummers te rapporteren welke 2 keer of meer gebruikt worden, wordt onderstaande SQL-opdracht uitgevoerd :

```
SELECT     IDNR, COUNT(IDNR) ;
FROM       WERKN ;
GROUP BY   IDNR ;
HAVING     COUNT(IDNR) >= 2 ;
INTO       TABLE DUBBEL.DBF
```

Tabel DUBBEL.DBF kan vervolgens gekoppeld worden aan tabel WERKN :

```
SELECT     WERKN.IDNR, WERKN.NAAM
FROM       WERKN, DUBBEL ;
WHERE      WERKN.IDNR = DUBBEL.IDNR ;
INTO       TABLE DUB_IDNR.DBF
```

5.2.3.4. Left outer join

```
SELECT     WERKN.IDNR, WERKN.NAAM, WERKN.AFDCODE,
           AFDELING.AFDNAAM ;
FROM       WERKN, AFDELING ;
WHERE      WERKN.AFDCODE = AFDELING.AFDCODE ;
UNION ;
SELECT     WERKN.IDNR, WERKN.NAAM, WERKN.AFDCODE,
           'niet gekend' ;
FROM       WERKN ;
WHERE      AFDCODE NOT IN
           (SELECT AFDCODE FROM AFDELING) ;
INTO       TABLE LEFTOUT.DBF
```

5.2.3.5. Right outer join

```
SELECT     WERKN.IDNR, WERKN.NAAM, WERKN.AFDCODE,
           AFDELING.AFDNAAM ;
FROM       WERKN, AFDELING ;
WHERE      WERKN.AFDCODE = AFDELING.AFDCODE ;
UNION ;
SELECT     0, 'niet gekend', '???', AFDELING.AFDNAAM ;
FROM       AFDELING ;
WHERE      AFDCODE NOT IN
           (SELECT AFDCODE FROM WERKN) ;
INTO       TABLE RIGHTOUT.DBF
```

5.2.3.6. Full outer join

```
SELECT     WERKN.IDNR, WERKN.NAAM, WERKN.AFDCODE,
           AFDELING.AFDNAAM ;
FROM       WERKN, AFDELING ;
WHERE      WERKN.AFDCODE = AFDELING.AFDCODE ;
UNION ;
SELECT     WERKN.IDNR, WERKN.NAAM, WERKN.AFDCODE,
           'niet gekend' ;
FROM       WERKN ;
WHERE      AFDCODE NOT IN
           (SELECT AFDCODE FROM AFDELING) ;
UNION ;
```

```
SELECT     0, 'niet gekend', '???', AFDELING.AFDNAAM ;
FROM       AFDELING ;
WHERE      AFDCODE NOT IN
           (SELECT AFDCODE FROM WERKN) ;
INTO       TABLE FULLOUT.DBF
```

5.2.3.7. SELECT DISTINCT

Door het specifiëren van DISTINCT in de SELECT-component worden identieke rijen verwijderd en komen ze slechts één keer voor in het resultaat (DISTINCT heeft betrekking op de gehele rij en niet alleen op de expressie die direct volgt op DISTINCT).

5.2.3.7.1. SELECT DISTINCT met inner join

```
SELECT     DISTINCT WERKN.WOONPL ;
FROM       WERKN, AFDELING ;
WHERE      WERKN.AFDCODE = AFDELING.AFDCODE ;
INTO       TABLE JOINDIST.DBF
```

5.2.3.7.2. SELECT DISTINCT met subquery met IN

```
SELECT     DISTINCT WOONPL ;
FROM       WERKN ;
WHERE      AFDCODE IN (SELECT AFDCODE FROM AFDELING) ;
INTO       TABLE SUBDIST.DBF
```

5.2.3.8. SUBQUERY

Met een subquery kunnen we slechts gegevens uit één tabel tonen, maar wel zo dat gegevens uit andere tabellen in de voorwaarden worden gebruikt. De gegevens die als voorwaarde moeten gelden, worden opgevraagd met behulp van een query. Een query die als voorwaarde gebruikt wordt in een query wordt een *subquery* genoemd (een onderliggende query).

Voorbeeld : toon NAAM en SALARIS uit de tabel WERKN voor de records waarvoor geldt dat het salaris lager ligt dan het maximum salaris. Door volgende query voert SQL eerst de eerste query uit. Vervolgens controleert SQL voor elke uitkomst van de hoofdquery of deze voldoet aan de eisen die gesteld zijn in de subquery. Omdat SQL hier tweemaal dezelfde tabel doorzoekt, hoeft de koppeling in dit geval niet noodzakelijkerwijs via een sleutel plaats te vinden.

```
SELECT     NAAM, SALARIS ;
FROM       WERKN ;
WHERE      SALARIS <
           (SELECT MAX(SALARIS) FROM WERKN)
```

Met de operator EXISTS vragen we aan SQL om voor elke uitkomst van de hoofdquery te controleren of deze ook voorkomt in de subquery. Beide onderstaande query's leveren hetzelfde resultaat op :

```
SELECT     AFDCODE,NAAM ;
FROM       WERKN ;
WHERE      WERKNR.AFDCODE IN
           (SELECT AFDCODE FROM AFDELING)
```

```
SELECT     AFDCODE,NAAM ;
FROM       WERKN ;
WHERE      EXISTS
           (SELECT *
            FROM AFDELING
            WHERE WERKN.AFDCODE = AFDELING.AFDCODE)
```

Bovenstaande query wordt een *gecorreleerde subquery* genoemd omdat een kolom wordt gebruikt die tot een tabel behoort die in een ander select-blok gespecifieerd is.

Voorbeeld : geef het IDNR van het record uit de tabel WERKN met het hoogste salaris

```
SELECT     IDNR ;
FROM       WERKN W1 ;
WHERE      W1.salaris > ANY
           (SELECT   SALARIS
            FROM WERKN W2)
```

Voorbeeld : geef IDNR van de records uit tabel WERKN waarvoor geldt dat het salaris tot de drie hoogste behoort uit de tabel WERKN

```
SELECT     IDNR ;
FROM       WERKN AS W1 ;
WHERE      3 >
           (SELECT COUNT(*)
            FROM WERKN AS W2
            WHERE W1.SALARIS < W2.SALARIS)
AND        IDNR IS NOT NULL
```

De subquery telt voor elk record het aantal records uit de tabel WERKN met een hoger salaris. Als dit aantal kleiner dan of gelijk is aan drie, wordt het IDNR van het records in het eindresultaat opgenomen. De additionele conditie IDNR IS NOT NULL is noodzakelijk omdat anders records zonder IDNR ook in het eindresultaat opgenomen worden. Nu worden ze verwijderd. Voorwaarde voor het uitvoeren van bovenvermelde query is echter dat er geen dubbele waarden voorkomen in de kolom salaris.

Voorbeeld : geef IDNR van de records uit tabel WERKN waarvoor geldt dat deze gewerkt hebben in afdelingen (kolom AFDCODE uit tabel AFDELING) waarvoor IDNR 57 niet gewerkt heeft

```
SELECT     IDNR ;
FROM       WERKN ;
WHERE      AFDCODE IN
           (SELECT AFDCODE
            FROM AFDELING
            WHERE AFDCODE NOT IN
             (SELECT AFDCODE
              FROM AFDELING
              WHERE IDNR=57))
```

Voorbeeld : geef IDNR van de records uit tabel WERKN waarvoor geldt dat het aantal bevorderingen (tabel BEVORDER) groter is dan het aantal afdelingen (tabel AFDELING) waarvoor ze gewerkt hebben

```
SELECT     IDNR ;
FROM       WERKN T1 ;
WHERE      (SELECT COUNT(*)
            FROM BEVORDER AS T2
            WHERE T1.IDNR = T2.IDNR)
           >
           (SELECT COUNT(*)
            FROM AFDELING AS T3
            WHERE T1.IDNR = T3.IDNR)
```

Voorbeeld : geef IDNR van de records uit tabel WERKNR waarvoor geldt dat deze in alle afdelingen gewerkt hebben

```
SELECT     IDNR ;
FROM       WERKNR ;
WHERE      NOT EXISTS
           (SELECT *
            FROM AFDELING
            WHERE NOT EXISTS
             (SELECT *
              FROM AFDELING
              WHERE IDNR = WERKN.IDNR))
```

Voorbeeld : geef IDNR en NAAM van de werknemers met een salaris groter dan het gemiddelde salaris van de afdeling waarvoor ze werken

```
SELECT     IDNR,NAAM ;
FROM       WERKN W1 ;
WHERE      SALARIS >
           (SELECT AVG(SALARIS)
            FROM WERKN W2
            WHERE W1.AFDCODE = W2.AFDCODE)
```

Voorbeeld : Geef het gemiddelde salaris van de werknemers met een anciënniteit (kolom ANCIEN) van meer dan 10 jaar op voorwaarde dat deze groep uit minstens 20 leden bestaat

```
SELECT     AVG(SALARIS) ;
FROM       WERKN W1 ;
WHERE      ANCIEN > 10 ;
GROUP BY   SALARIS ;
HAVING     COUNT(*) >= 20
```

5.3. Data-administratie en database-administratie

Gegevens zijn voor een organisatie heel belangrijk en daarom moeten organisaties regels en procedures formuleren voor de bescherming en het juiste gebruik van hun gegevens. *Data-administratie* en *database-administratie* zijn twee functies binnen een organisatie die de verantwoordelijkheid hiervoor dragen. We beginnen met het onderbouwen van de noodzaak van data-administratie : het opstellen en onderhouden van een overzicht van de gegevens binnen de gehele organisatie. Vervolgens behandelen we de database-administratie. Deze functie heeft betrekking op een specifieke database en niet op de organisatie als geheel. Database-admininistratie slaat op het beheer van de databasestructuur, de databasebewerkingsactiviteiten en het DBMS zelf.

5.3.1. Gegevens als hulpbron voor de organisatie

De gegevens van een organisatie zijn evenzeer een hulpbron als de gebouwen, de machines en de financiële activa. Het verzamelen van gegevens kost tijd en geld, en gegevens spelen niet alleen een belangrijke rol bij het besturen van operationele en administratieve processen, maar ook bij de controle van de kwaliteit van de producten en dienstverlening. Gegevens kunnen er vaak toe bijdragen dat een onderneming zijn concurrentiepositie kan handhaven of versterken. Denk bijvoorbeeld aan de waarde van een klantenlijst voor een postorderbedrijf.

Organisatiegegevens moeten vanwege hun waarde net als andere hulpbronnen worden beheerd en bewaakt. Daarom hebben veel ondernemingen dan ook data-administratie- en database-administratie-afdelingen ingericht. Zij moeten zorgen voor de gegevensbeveiliging en effectief gegevensgebruik mogelijk maken.

5.3.2. Data-administratie

We maken het onderscheid tussen data-administratie en database-administratie op basis van het volgende : data-administratie is een functie met als gezichtsveld de gehele onderneming/organisatie en database-administratie richt zich op een specifieke database.

Om de noodzaak van data-administratie duidelijk te maken, maken we een vergelijking met een universiteitsbibliotheek. Zo'n bibliotheek is slechts nuttig als de boeken, rapporten en tijdschriften die erin bijgehouden worden door mensen benaderbaar zijn. De bibliotheek moet zijn inventaris/catalogus op de ene of andere manier beschrijven zodat potentiële gebruikers aan de weet kunnen komen wat er beschikbaar is. Data-administratie van een onderneming is nog moeilijker dan in het geval van een universiteitsbibliotheek. Gegevens van de organisatie kunnen behalve in boeken en rapporten ook opgenomen zijn in documenten, rekenbladen, grafieken, tekeningen, e.d. Hoe moeten we dit alles beschrijven ? Welke zijn de elementaire gegevenscategorieën ? De antwoorden op deze vragen zijn belangrijk omdat zij mede bepalen hoe gegevens moeten worden georganiseerd, beheerd, bewaakt en benaderd.

Data-administratie is geen eenvoudige activiteit. Men moet gegevens bewaken en beschermen en tegelijkertijd de gebruikswaarde vergroten. Om

dit te bewerkstelligen moeten een aantal activiteiten worden uitgevoerd. Hieronder volgt een overzicht van de *functies van data-administratie*.

Marketing

- het bestaan van data-administratie bekend maken bij de rest van de organisatie
- de diensten die de data-administratie kan verlenen kenbaar maken

Datastandaarden

- opstellen van standaarden voor het beschrijven van gegevens

Dataprocedures

- opstellen van procedures voor de hele organisatie voor het omgaan met gegevens (beveiliging en beschikbaarstelling)

Bemiddeling bij dataconflicten

- opstellen van procedures voor het rapporteren van conflicten ten aanzien van gegevensgebruik
- beschikken over de bevoegdheid maatregelen te nemen die tot de oplossing van het conflict leiden

Beheer van de investering in gegevens

- de aandacht vestigen op de waarde van de investering in gegevens
- nieuwe methoden en technieken onderzoeken
- vooruitlopen op informatiebehoeften, i.p.v. slechts op gestelde vragen te reageren

5.3.3. Database-administratie

De database-administratie beperkt haar aandacht tot een specifieke database en tot de systemen die daar bewerkingen op uitvoeren. De database-administratie (DBA) is actief binnen het raamwerk dat de data-administratie heeft gecreëerd om de ontwikkeling en het gebruik van databases te bevorderen. De database-administratie voor individuele of persoonlijke databases is aanzienlijk eenvoudiger dan die voor groeps- of afdelingsdatabases en deze laatste is weer eenvoudiger dan die voor organisatiedatabases. Database-administratie voor persoonlijke databases is vaak niet geformaliseerd. Gebruikers zorgen zelf voor backups en de documentatie is beperkt. De gebruiker vervult zelf de DBA-functies.

Bij groepstoepassingen moet meer administratie verricht worden. Veelal verzorgen een of twee leden van de groep in een deel van hun werktijd deze functie. Bij een database op organisatieniveau neemt DBA vaak zoveel tijd in beslag dat zelfs een enkele voltijds DBA-functie onvoldoende is.

Het is de verantwoordelijkheid van de database-administrator om de ontwikkeling en het gebruik van een database te bevorderen binnen het kader van de richtlijnen die door de data-administratie werden opgesteld. Meestal betekent dit dat de database-administrator een compromis moet zoeken tussen de tegenstrijdige belangen van databasebeveiliging en maximale beschikbaarheid van gegevens voor de gebruikers. Zoals in het geval van data-administratie, kunnen ook m.b.t. database-administratie verschillende functies (taken voor de database-administrator) onderscheiden worden.

Beheer van de databasestructuur

De database-administrator dient in een vroeg stadium betrokken te worden bij de ontwikkeling van de database, deelnemen aan het opstellen van de functionele specificaties, en een stem hebben in de keuze van een DBMS. De database-administrator moet eveneens betrokken zijn bij het opstellen van procedures ter verzekering van de integriteit van de databasegegevens. Een doelmatige database-administratie houdt in dat er procedures beschikbaar zijn voor het registreren van gebruikerswensen t.a.v. veranderingen in de database ; dit moet op een zodanige wijze gebeuren dat alle betrokkenen de gewenste veranderingen kunnen beoordelen, opdat vervolgens een gefundeerde beslissing kan worden genomen over het al of niet doorvoeren van de voorgestelde verandering. Wegens de omvang en de complexiteit van een database en zijn toepassingen, kunnen wijzigingen soms onverwachte effecten hebben. De database-administrator is ook verantwoordelijk voor het herstel van de database in dergelijke gevallen en voor het verzamelen van informatie over de oorzaak van het probleem en over de wijze waarop het opgelost kan worden. Ook het aanleggen van documentatie over de databasestructuur behoort tot de taken van de database-administrator.

Beheer van gegevensverwerking

De DBA beschermt de gegevens maar verwerkt deze niet zelf, want de DBA is geen gebruiker van het systeem ; de DBA is geen beheerder van gegevenswaarden maar alleen van de gegevensverwerking. De database is een gemeenschappelijk gebruikt hulpmiddel en de DBA voorziet in standaarden, richtlijnen, controleprocedures en documentatie ("data-dictionary") om een goede samenwerking tussen gebruikers te bevorderen. Een belangrijk aspect van de gegevensverwerking is het vastleggen van gegevensverantwoordelijkheid en het beheer van de toegangs- en wijzigbevoegdheden. Omdat gegevens gemeenschappelijk gebruikt worden,

kunnen conflicten ontstaan t.a.v. de gebruikersbevoegdheden. De DBA stelt samen met de gegevensverantwoordelijken de rechten van de gebruikers vast (cfr. supra : betrouwbaarheid van de database en databasebeveiliging). Belangrijk bij gegevensverwerking is verder het ontwikkelen en documenteren van backup- en recovery-procedures en het opleiden van medewerkers in het gebruik ervan.

Beheer van het DBMS

Behalve het beheren van de databasestructuur en de gegevensverwerkingsactiviteiten, moet de DBA ook het DBMS zelf beheren, hetgeen o.m. inhoudt dat de DBA het gebruik van de database beoordeelt. Veel DBMS-producten voorzien in de mogelijkheid om statistische gegevens over het databasegebruik te genereren. Zo kan men zien welke gebruikers actief zijn geweest, welke bestanden zijn benaderd en welke benaderingsmethoden daarbij zijn gebruikt. Ook foutrapporten kunnen gegenereerd worden ; deze gegevens kan de DBA o.m. gebruiken om te beslissen of er wijzigingen in het database-ontwerp moeten worden aangebracht die de verwerkingssnelheid kunnen vergroten of de taken van gebruikers kunnen vereenvoudigen (afstellen of tunen van het systeem). Als de leverancier van het DBMS een nieuwe versie van het product aankondigt, dan moet de DBA dit beoordelen in het licht van de bestaande gebruikersbehoeften. Als nieuwe functies worden toegevoegd, moeten gebruikers hiervan op de hoogte worden gesteld en instructie krijgen in het gebruik ervan het DBMS-product moet ook worden afgesteld en aangepast aan systeemsoftware (het besturingssysteem).

Een aantal evoluties heeft de huidige populariteit van het gebruik van marketing databases versneld : de reclame- en marketingkosten stijgen voortdurend, de markt is versnipperd in talloze deelmarkten, de merkentrouw daalt, nieuwe winkel- en betaalgewoonten ontstaan, informaticamoelijkheden nemen toe, de klant wordt mondiger en veeleisender, enz.

In principe is de marketing database een verdere ontwikkeling van het adressenbestand. Een *adressenbestand* is een verzameling van gegevens over klanten, prospects of suspects : hun namen, adressen, telefoonnummers, enz. Een *database* is echter meer dan een loutere verzameling van namen en adressen. Een database bewaart ook informatie over de contacten tussen de eigen onderneming en klanten, prospecten of suspecten. Verkopen, afgesloten contracten, respons op een direct mail-actie, bezoeken van een verkoopteam, resultaten van een telemarketing-actie of ieder ander relevant type contact tussen de twee partijen is geregistreerd in een database. Het gaat hier zowel om *acties* van de onderneming naar de markt toe als om reacties van de markt naar de onderneming toe. Bovendien houdt de database bij welke producten, afdelingen of verkopers betrokken zijn bij een

bepaalde actie : wie koopt welke producten, hoe frequent worden die producten aangeschaft, enz. Dit laatste type gegevens noemen we ondernemingsgegevens.

De *voordelen* van het gebruik van marketing databases zijn uiteenlopend. Door gegevens in de database te analyseren, heeft men een krachtig on-line onderzoeksinstrument ter beschikking om te ontdekken wie producten of diensten koopt. Hierdoor kunnen klantprofielen opgesteld worden. Een database laat derhalve toe om segmenten binnen het klantenbestand te identificeren en om prospects, gekenmerkt door het profiel van de eigen klanten of die van de concurrentie, op een meer efficiënte en effectieve manier te bereiken. Het spaart geld uit, verhoogt de respons en versterkt de relaties met de klanten en prospects door hun individualiteit te erkennen. Ook de verkoopploeg profiteert mee van het bestaan van een marketing database. Het is immers van cruciaal belang om de beschikbare tijd van de verkoopploeg zo optimaal mogelijk aan te wenden. Een database kan verkopers helpen om de meest beloftevolle prospects te identificeren. Door een score toe te kennen aan elke prospect en alleen de meest interessante onder hen te contacteren, zal de respons gevoelig hoger liggen. Een database houdt informatie bij over klanten of prospects over een bepaalde tijdsperiode. Dit betekent dat evoluties in het klantenbestand kunnen waargenomen worden m.b.t. koopvolume, aankoopfrequentie of respons of mailings. Dit laat toe om specifieke acties te ontwerpen : speciale behandelingen voor de beste of minder winstgevende klanten, het reactiveren van verloren klanten, het upgraden van klanten, enz. Een database kan bovendien de kans inschatten dat een bepaalde klant zijn relatie met de onderneming zal stopzetten. Gemiddeld genomen kost het vijf tot tien keer meer om een nieuwe klant binnen te halen dan om een bestaande klant te behouden. Het is dan ook nuttig om die klanten die de kans vertonen contact met de onderneming te verliezen, in het oog te houden. Het opbouwen van een langdurige relatie met klanten en het verbeteren van de service naar hen toe is van essentieel belang. Door de reacties van klanten nauw in de gaten te houden, krijgen we ook meer zicht op hetgeen werkt en wat niet werkt. Dit kan opportuniteiten openen voor nieuwe producten of diensten.

Omwille van het belang van databases voor de onderneming, heeft zich de afgelopen jaren een belangrijke verschuiving voltrokken van transactionele (*OLTP*) naar analytische systemen (*OLAP*). Deze verschuiving, die o.m. het gevolg is van zowel het toenemend belang van databases ter ondersteuning van beslissingsprocessen ("decision support"), ligt mede aan de basis van een ommekeer die tegenwoordig geïllustreerd wordt door het groeiende succes van *data warehousing* en *data mining*.

NOTEN

[1] Voor een introductie over data mining, zie http://www. rpi.edu/~arunmk/dml. html.

[2] Zie hierover ; HEIRMAN, J. Marketing information-systemen, m Van Vooren, E. (red.), Direct Marketing Inspiratieboek, Tielt, Lannoo, 1996, pp. 67 e.v.

[3] Zie o.m. Van den Haspel, T., Client sever management, Schoonhoven, Academic Service, 1993.

[4] Zie hierover Van Der Lans, R.F., Het relationele model voor database management systemen, Schoonhoven, Academic Service, 1988, pp. 179 e.v.

[5] CODD, E., A relational model of data for large shared data banks, Communications of the ACM, Jrg. 13, nr. 6, 1970, pp. 377-387.

[6] FAGIN, R., A normal form for relational databases that is based on domains and keys, AC Transactions on Database Systems, Jrg. 6, nr. 3, 1981, pp . 387-415.

[7] CHEN, P.P., The entity-relationship model- toward a unified view of data, ACM Transactions on Database Systems, Jrg. 1, nr. 1, 1976, pp. 9-36.

INHOUDSOPGAVE

INLEIDING 2

1. Marketinginformatie-systemen 3
1.1.Transactionele (data-inzameling) toepassingen 6
1.2. Marketing Action Support Systems (MASS) 6
1.3. Marketing Decision Support Systems (MDSS) 7

2. Architecturen vooor multi-user databasetoepassingen 9
2.1. Teleprocessing-systemen 11
2.2. Resource sharing-systemen 12
2.3. Client/server-archtecturen 15

HET RELATIONELE MODEL EN NORMALISATIE 20

1. Basisbegrippen 20
1.1. Bestandsverwerkingssystemen 21
1.2. Databaseverwerkingssystemen 23

2. Definitie van het begrip database 26

3. Database-terminologie 27

4. Ontwerp en opzet van databases 30
4.1. Toepassingsdatabases 31
4.2. Onderwerpdatabases 31
4.3. Logisch database-ontwerp 33
4.3.1. Relaties 34
4.3.2. Normalisatie 36
4.3.2.1. Normaalvormen 38
4.3.2.1.1. De eerste normaalvorm 38
4.3.2.1.2. De tweede normaalvorm 38
4.3.2.1.3. De derde normaalvorm 40
4.3.2.1.4. De Boyce-Codd normaalvorm 40
4.3.2.1.5. De vierde normaalvorm 42
4.3.2.1.6. De domein/sleutel-normaalvorm 43
4.3.2.1.6.1. DS/NV : voorbeeld 1 44
4.3.2.1.6.2. DS/NV : voorbeeld 2 45
4.3.2.2. CASE-tools 46
4.4. Basisbeginselen bij de opzet van databases 50
4.4.1. Het definiëren van de databasestructuur in het DBMS 50
4.4.1.1. Primaire sleutels 53
4.4.1.2. Alternatieve sleutels 54

4.4.1.3. Refererende sleutels 54
4.4.2. Reservering van opslagruimte 59
4.4.3. Het vullen van de database met gegevens 59
4.5. Relationele gegevensmanipulatie 59
4.5.1. Hulpmiddelen voor verwerkingsbesturing 60
4.5.2. Beveiliging en integriteit 61
4.5.3. Database-applicaties 61
4.5.3.1. Opbect- en viewmaterialisatie 62
4.5.3.2. Formulieren 64
4.5.3.3. Rapporten 65
4.5.3.4. Relationele datamanipulatietalen 65
4.5.3.5. Interfaces met het DBMS 66
4.5.3.6. Open Database Connectivity (ODBC) 69

5. Structured Query Language 71
5.1. Componenten van de SELECT-instructie 71
5.2. Relaties tussen tabellen 73
5.2.1. Soorten joins 74
5.2.2. Relaties tussen joinkolommen 75
5.2.2.1. De populaties van de joinkolommen zijn hetzelfde 76
5.2.2.2. De populatie van een joinkolom is een deelverzameling 77
5.2.2.3. De populaties van de joinkolommen zijn conjunct 79
5.2.2.4. De populaties van de joinkolommen zijn disjunct 80
5.2.3. Uitbreiding 81
5.2.3.1. Inner join 81
5.2.3.2. Autojoin 81
5.2.3.3. Tellen van dubbels 82
5.2.3.4. Left outer join 83
5.2.3.5. Right outer join 83
5.2.3.6. Full outer join 83
5.2.3.7. SELECT DISTINCT 84
5.2.3.7.1. SELECT DISTINCT met inner join 84
5.2.3.7.2. SELECT DISTINCT met subquery met IN 84
5.2.3.8. Subquery 85
5.3. Data-administratie en database-administratie 88
5.3.1. Gegevens als hulpbron voor de organisatie 89
5.3.2. Data-administratie 89
5.3.3. Database-administratie 90

NOTEN 94

INHOUDSOPGAVE 95