III. MANIPULATIETAAL

III.1. OVERZICHT

De manipulatie op de gegevens die behoren tot de typen van een bepaald conceptueel model en zijn opgeslagen in de database verloopt via een zogenaamde manipulatietaal. Deze taal bestaat uit een aantal opdrachten. Deze opdrachten worden in drie categorieën verdeeld:

1) Opvragingsopdrachten,
2) Uitbreidingsopdrachten en
3) Mutatie-opdrachten.

Een query bestaat uit een aantal opdrachten. Hierbij is het toegestaan om meer dan 1 opvragingopdrachten te gebruiken. Het resultaat van een query kan dus uit meerdere delen bestaan.

Hieronder volgt een summiere beschrijving van deze categorieën, tezamen met een opsomming van de opdrachten die in deze categorieën vallen.

Opvragingsopdrachten kunnen gebruikt worden om gegevens die in de database zijn opgeslagen op te vragen. De opvragingsopdrachten zijn:

- de **get**-opdracht.
  Deze opdracht kan gebruikt worden om gegevens van typen, behorende tot een bepaalde database, op te vragen.

- de **value**-opdracht **zonder** value-specificatie.
  Via deze opdracht kunnen waarden van de met behulp van de ’value-opdracht met value-specificatie’ gedefinieerde variabelen worden opgevraagd.

Uitbreidingsopdrachten kunnen gebruikt worden om bepaalde afgeleide gegevens tijdelijk aan de in de database opgeslagen verzameling gegevens toe te voegen. De uitbreidings-opdrachten zijn:

- de **value**-opdracht **met** value-specificatie.
  Middels deze opdracht kunnen we een tijdelijke variabele definiëren en tevens van een waarde voorzien. Bij volgende opdrachten kan zo’n variabele gebruikt worden in expressies.

- de **extend**-opdracht.
  Deze opdracht dient om aan een samengesteld type een bepaald attribuut tijdelijk toe

te voegen. De waarde van het toegevoegde attribuut wordt daarbij afgeleid uit reeds aanwezige gegevens. Zo'n toegevoegd attribuut wordt een virtueel attribuut genoemd. Een virtueel attribuut kan natuurlijk bij volgende opdrachten gebruikt worden.

Mutatie-opdrachten kunnen gebruikt worden om de database uit te breiden met nieuwe gegevens (instances), bepaalde instances te verwijderen of om de in de database opgeslagen gegevens te veranderen. De integriteit van de database blijft hierbij altijd gewaarborgd. De mutatie-opdrachten zijn:

- de **insert**-opdracht.
  Deze opdracht kan gebruikt worden om een instance van een samengesteld type aan de database toe te voegen. De verschillende attributen van dit type worden hierbij altijd van een waarde voorzien.

- de **delete**-opdracht.
  Deze opdracht kan gebruikt worden om instances van een samengesteld type uit de database te verwijderen. Een opdracht tot verwijdering zal alleen uitgevoerd worden als hierdoor de integriteit van de database niet aangetast wordt.

- de **update**-opdracht.
  Met behulp van deze opdracht kunnen de attributen van instances van een samengesteld type van een andere waarde worden voorzien.

- de **cascade**-opdracht.
  Met behulp van deze opdracht kunnen recursieve queries geformuleerd worden. De attributen van instances van een samengesteld type kunnen, in een door de opdracht bepaalde volgorde, herhaaldelijk van een andere waarde worden voorzien.

Naast de hierboven gespecificeerde opdrachten zijn er nog drie opdrachten. Deze zijn voor het opnemen van extra regels in de uitvoer (de **newline**-opdracht), opnemen van commentaar in een query (de **#**-opdracht) en het weergeven van een tekst **tijdens de verwerking** (let op: **niet** in het resultaat) van de query (de **echo**-opdracht). Deze opdrachten maken geen gebruik van gegevens in de database en behoren niet tot de manipulatietaal. Ze zijn aanwezig om queries te verduidelijken en gebruiksvriendelijker te maken. Ze worden aan het eind van dit hoofdstuk beschreven.

De beschrijving van deze opdrachten vangt aan met een beschrijving van de get-opdracht. In deze beschrijving zullen facetten aan de orde komen die ook voor de overige opdrachten van belang zijn. Het is daarom aanbevelenswaard eerst de verhandelingen betreffende deze opdracht in z'n totaliteit te doorlopen alvorens tot andere opdrachten over te gaan.

Bij de behandeling van de verschillende opdrachten zal veelvuldig gebruik gemaakt worden van voorbeelden. Deze voorbeelden hebben betrekking op de warenhuis-database. Het conceptuele model van deze database is gegeven in X.1. (APPENDICES).

Alvorens over te gaan tot de beschrijving van bovengenoemde opdrachten worden de formaten van de grootheden behandeld, omdat ze binnen vrijwel iedere opdracht gebruikt worden.

III.2. FORMATEN VAN CONSTANTEN

De attributen waarmee gewerkt wordt hebben allen een bepaald domein. Het domein van een attribuut kan zijn: INTEGER, REAL, ALFANUMERIEK, BOOLEAN of DATE.

Het zal blijken dat het ook wenselijk is met constanten te kunnen werken. Afhankelijk van het betreffende domein moeten attributen en constanten het volgende formaat hebben:

- Attributen met domein INTEGER en REAL.
  Bij deze typen is er sprake van getallen. Deze getallen hebben het gebruikelijke formaat. Enkele voorbeelden:
  - INTEGER-getallen: 2, 123 en -89 (gehele getallen).
  - REAL-getallen: 12.3, 0.29, 123 en -0.01 (gebroken getallen met decimale punt).

- Attributen met domein ALFANUMERIEK.
  Bij alfanumeriek is er sprake van karakterstrings. Deze strings kunnen we gebruiken om bijvoorbeeld de naam van een persoon weer te geven. Strings moeten altijd tussen dubbele quotes geplaatst worden. Bijvoorbeeld: "Jansen", "Pieterse" maar ook: "Tuinstraat 33" en "REF. 120893/MD/v1". Een dubbele quote in een string moet direct gevolgd worden door een extra dubbele quote: "citaat ""BLa Bla bla"".".

- Attributen met domein BOOLEAN.
  Dit zijn de zogenaamde logische waarden. Een constante met dit domein heeft als waarde TRUE of FALSE eventueel in kleine letters. Andere waarden dan TRUE, true, FALSE of false zijn niet toegestaan.

- Attributen met domein DATE.
  Dit zijn geldige kalenderdatums volgens de interpretatie jjjjmmdd tussen 19011214 en 20380118. Met een datum kan alleen met behulp van datum-functies gerekend worden.

III.3. GET-OPDRACHT

III.3.1. Inleiding

De letterlijke vertaling van ’get’ uit het Engels is ’halen’. De get-opdracht kan dan ook gebruikt worden om bepaalde gegevens uit de database op te halen. Hierbij blijven de waarden van in de database opgeslagen gegevens onveranderd.

Met de get-opdracht kan men alleen de gegevens van samengestelde typen opvragen. De

eenvoudigste get-opdracht is die waarbij alle instances van een samengesteld type worden opgevraagd, bijvoorbeeld:

get werknemer.

Haal alle gegevens van alle werknemers.

Merk op dat opdrachten altijd met een **punt** moeten worden afgesloten. Het resultaat van deze opdracht is een tabel met alle gegevens van alle werknemers. Van iedere werknemer wordt de betreffende instance identificatie met alle bijbehorende attributen gegeven:

werk-nemer	naam	voorl	adres	postc	woonplaats	afdeling	funktie
WN1	Smits	P.J.	Torenweg 12	2626AD	Delft	P&O	boekh.
WN2	Peters	W.	Peensteeg 7	2610GK	Delft	Verkoop	1-verk.
WN3	Zomers	F.	Mekelweg 234	2628DK	Delft	Inkoop	inkoper

We hebben nu dus een tabel gekregen met van iedere werknemer de identificatie (WN1 enz.) en de waarden van alle attributen. Voor veel toepassingen zal deze mogelijkheid om van een bepaald samengesteld type van alle instances alle attributen te krijgen niet afdoende zijn. De get-opdracht kan worden uitgebreid met de mogelijkheid om:

- te kunnen **selecteren op attributen**. Men is bijvoorbeeld alleen geïnteresseerd in de naam en het adres van de werknemers. Hiertoe bestaat er binnen de get-opdracht een mogelijkheid tot selectie van attributen. Deze selectie is niet beperkt tot de attributen van het type zelf maar kan ook attributen van direct gerelateerde typen (de ontledingsboom) bevatten.

- te kunnen **selecteren op** uit de database te halen **instances**. Men is bijvoorbeeld alleen geïnteresseerd in de gegevens van de werknemers van een bepaalde afdeling. Hiertoe bestaat er binnen de get-opdracht een mogelijkheid tot selectie van instances.

- bepaalde **berekeningen** uit te voeren op attributen die bij de get-opdracht van belang zijn. Deze berekeningen kunnen alleen op attributen met domein REAL of INTEGER, getallen dus, uitgevoerd worden. Deze berekeningen kunnen bij bovengenoemde twee vormen van selectie een rol spelen:
  - ten eerste **bij** de vorm waarin bepaalde **geselecteerde attributen** in het resultaat van de get-opdracht moeten worden geplaatst. Stel bijvoorbeeld dat men de prijs van een bepaald artikel wil weten om hierop een bepaalde korting te geven. Natuurlijk zou men gewoon de prijs van het artikel op kunnen vragen om dan vervolgens zelf de korting hier vanaf te halen. Maar waarom zou men dit zelf doen indien het ook mogelijk is dit met behulp van een bepaalde afbeelding van geselecteerde attributen te doen.

  - ten tweede **bij de selectie van instances**. Men wil bijvoorbeeld alleen gegevens hebben over artikelen waarbij de korting meer dan fl. 100,- bedraagt. Om dit soort afbeeldingen van attributen mogelijk te maken bestaat er de mogelijkheid om deze te gebruiken in expressies.

- bepaalde nieuwe **geaggregeerde informatie** op grond van gegevens opgeslagen in de database af te leiden. Men moet hier denken aan bijvoorbeeld het berekenen van het totaal aan uitstaande vorderingen, het bepalen van het aantal werknemers dat op een bepaalde afdeling werkt en dergelijke. Hiertoe moet het mogelijk zijn bepaalde functies op een verzameling van geselecteerde gegevens toe te passen. Deze functies worden **set-functies** genoemd.

Deze opsomming is niet bedoeld om direct al een totaaloverzicht te geven van de verschillende opvragingsmogelijkheden van de get-opdracht, maar eerder om te komen tot een logische opdeling van de verschillende structuren die in de get-opdracht mogelijk zijn. Deze structuren en het nut ervan zullen na het doornemen van de volgende paragrafen duidelijk zijn.

III.3.2. Selectie van attributen

In de inleiding is een voorbeeld gegeven van een get-opdracht waarbij van een bepaald samengesteld type alle instances van dit type met alle attributen uit de database gehaald werden. In dit hoofdstuk zal aan de orde komen welke mogelijkheden er geboden worden om te selecteren op instances en attributen en hoe men dat specifeert.
Stel dat men, in tegenstelling tot het in de inleiding gegeven voorbeeld, alleen geïnteresseerd is in de namen van de werknemers, dan kan men dit als volgt specificeren:

get werknemer **its** naam.

Haal de namen van alle werknemers.

Het resultaat bestaat ook nu uit een tabel waarin alle werknemers voorkomen. Per werknemer wordt gegeven zijn identificatie, tezamen met enkel het gewenste attribuut ’naam’.

Het gedeelte van een opdracht dat aangeeft welke attributen geselecteerd moeten worden, in bovenstaande opdracht ’**its** naam’, wordt de **attribuut selectie specificatie** genoemd.

Met behulp van zo’n attribuut selectie specificatie kunnen we ieder gewenst attribuut selecteren. Indien men meer dan **één** attribuut wil zien, dan moet men de verschillende attribuutnamen door een komma (d.w.z. ’,’) scheiden. Bijvoorbeeld:

get werknemer **its** naam, voorl, adres, postcode, woonplaats.

Haal van iedere werknemer zijn volledige naam en adres.

Het is ook mogelijk met behulp van de get-opdracht waarden van attributen van onderliggende samengestelde typen op te vragen. Neem bijvoorbeeld nogmaals het type werknemer. Het type afdeling is voor dit type een onderliggend samengesteld type. Indien men met behulp van een get-opdracht gegevens wil opvragen over de afdelingen waarop werknemers werken dan kan men dit als volgt doen:

get werknemer **its** naam, afdeling.

Haal van alle werknemers hun naam en de afdeling waarop deze werknemer werkt.

Het resultaat is een tabel met daarin per werknemer vermeld de identificatie, naam en de afdeling waarop hij werkt. Deze afdeling waarop de werknemer werkt wordt in dit geval aangegeven door de identificatie van de betreffende afdeling. Dat dit niet de meest hanteerbare vorm is spreekt voor zich. Beter zou het zijn indien er een mogelijkheid was om bepaalde attributen van het type afdeling te specificeren. Men zou bijvoorbeeld kunnen wensen dat in plaats van de identificatie de naam van de betreffende afdeling gehaald wordt. De attributen van onderliggende samengestelde typen worden gespecificeerd door herhaald gebruik te maken van **its**. De gewenste afdelingsnaam kan men als volgt opvragen:

get werknemer **its** afdeling **its** afdnaam.

Het resultaat is nu een tabel van alle werknemers met per werknemer de instance identificatie en de naam van de afdeling waarop hij werkt. Indien men nu ook nog de naam van de werknemer wil hebben moet de opvraging als volgt geformuleerd worden:

get werknemer **its** naam, afdeling **its** afdnaam.

Haal van iedere werknemer zijn naam en de naam van de afdeling waarop hij werkt.

De eerste **'its'** werkt op alle door komma's gescheiden gespecificeerde attributen. De volgende get-opdracht is dus **niet correct**:

get werknemer **its** naam, **its** afdeling **its** afdnaam.

De tweede **'its'** is hierin overbodig.

Bovenstaand voorbeeld wil natuurlijk niet suggereren dat indien men een overzicht wil hebben met de namen van alle afdelingen, men dat via het type werknemer moet opvragen. In bovenstaand voorbeeld wil men namelijk steeds iets weten van bepaalde werknemers. Daarom zijn al deze opvragingen van de vorm:

get werknemer **its** ... .

Indien een overzicht van alle afdelingen gewenst is, is men kennelijk geïnteresseerd in het

type afdeling en niet in het type werknemer. De opvraging van alle namen van afdelingen verloopt dan ook als volgt:

get afdeling **its** afdnaam.

Haal de namen van alle afdelingen.

Het type waarin men geïnteresseerd is noemen we het **access-type (AT)**; het type van waaruit het conceptuele model benaderd wordt.
Met behulp van '**its**' kunnen dus de attributen van onderliggende samengestelde typen worden gespecificeerd. Door bij werknemer 'its' te gebruiken worden de attributen van werknemer gespecificeerd en door bij het attribuut afdeling van werknemer nogmaals 'its' te gebruiken kunnen de attributen van afdeling gespecificeerd worden. Afhankelijk van het conceptuele model kan men hiermee doorgaan. Neem bijvoorbeeld het type verkoop. De volgende opvraging betreffende dit type is correct:

get verkoop **its** verkart **its** artikel **its** soort **its** beschrijving.

Haal de omschrijving van alle werkelijk verkochte artikelen.

Het is niet uitgesloten dat in de tabel met verkregen beschrijvingen bepaalde beschrijvingen meerdere keren voorkomen.

Het blijkt dus dat met behulp van 'its'-en een bepaald pad naar beneden wordt gespecificeerd. Zo'n pad noemen we een **attribuutpad (AP)**. Immers, alles wat op zo'n pad ligt is in zekere zin attribuut van het access-type. Het is direct benaderbaar via de 'its'-specificatie.

Als laatste voorbeeld de volgende opvraging:

get verkoop **its**
datum,
klant **its** naam,
bedrag,
aanbet,
klant **its** schuld,
verkart **its** artikel **its** beschrijving,
hoeveelheid,
verkart **its** artikel **its** voorraad.

Haal een aantal gegevens op die bereikbaar zijn vanuit het access-type verkoop. Hierin zijn datum, klant, bedrag, aanbet, verkart en hoeveelheid attributen van verkoop.

Het blijkt dus dat een willekeurig aantal attribuutpaden in een willekeurige volgorde kan worden gespecificeerd. Het eerste type dat in deze paden genoemd wordt is steeds een

attribuut van het access-type. Het laatste attribuut is het attribuut waarvan de waarde gewenst wordt. Met behulp van attribuutpaden kan men ieder attribuut in de ontledingsboom van het access-type specificeren.

III.3.3. Selectie van instances

Tot dusver zijn voorbeelden getoond van de get-opdracht waarbij steeds alle instances van het betreffende samengestelde type geselecteerd werden. In deze paragraaf zal aan de orde komen welke mogelijkheden er geboden worden om te selecteren op instances.

Met behulp van de identificatie

De eenvoudigste vorm van selectie op instances is die waarbij gezocht wordt naar de gegevens van een bepaalde instance waarvan de identificatie bekend is. Stel bijvoorbeeld dat men de gegevens van de werknemer met identificatie WN9 wil weten. Deze kan men als volgt opvragen:

get werknemer "WN9".

Haal alle gegevens van de werknemer met identificatie WN9.

Het resultaat bestaat nu uit de instance-identificatie van de werknemer tezamen met alle bijbehorende attribuutwaarden. Omdat de identificatie van een instance uniek is kan men op deze manier de gegevens van ten hoogste **één** werknemer krijgen, althans indien er daadwerkelijk gegevens over de betreffende werknemer opgeslagen zijn.

Met behulp van een voorwaarde

Indien men de identificatie van de werknemer waarover men iets wil opvragen niet kent, moet men anders te werk gaan. Neem echter aan dat bekend is dat diens naam Pieterse is. De gewenste gegevens worden dan als volgt opgevraagd:

get werknemer
    where naam = "Pieterse".

Haal alle gegevens van alle werknemers met naam Pieterse.

Het gedeelte ’where naam = "Pieterse"’ specificeert hier welke instances van het type werknemer geselecteerd moeten worden. Dit gedeelte heet de **instance selectie specificatie**. Het gedeelte ’naam = "Pieterse"’ is een voorwaarde. "Pieterse" wordt in deze opdracht als een constante beschouwd en heeft het formaat van een constante met domein ALFANUMERIEK omdat het attribuut naam dit domein heeft. Men moet het formaat van de constanten die in opdrachten worden gebruikt altijd zodanig kiezen dat dit formaat overeenkomt met het domein van de attributen waarop gewerkt wordt. Bij constante mag overigens het aantal tekens kleiner

zijn dan in het domein aangegeven is.

Het resultaat van bovenstaande opvraging bestaat uit alle gegevens van minimaal nul werknemers. Immers, er kan een willekeurig aantal mensen bij het bedrijf werkzaam zijn luisterend naar de naam Pieterse. Indien men de gegevens van meer dan één Pieterse krijgt en de gegevens van slechts **één** bepaalde Pieterse wil hebben, dan moet men op meer attributen gaan selecteren, bijvoorbeeld ook op voorletters. Er zijn dus meer voorwaarden nodig. Hoe men dit moet formuleren zal in een volgende paragraaf behandeld worden.

Merk overigens het volgende op:

get artikel "7".
get soort "kleding".

Alhoewel artikel als domein INTEGER heeft en soort als domein ALFANUMERIEK worden de identificaties in deze get-opdrachten in beide gevallen tussen dubbele quotes geplaatst. Het formaat van deze instance-identificaties wordt echter verschillend indien men ze gebruikt in selectie met behulp van een voorwaarde, bijvoorbeeld:

get verkoop **where** verkart **its** artikel = 7.
get artikel **where** soort = "kleding".

Hier worden de beide identificaties namelijk als constanten beschouwd, zodat ze het formaat krijgen dat overeenkomt met het domein van de identificaties van de betreffende typen.

Met behulp van wildcards in voorwaarden

Bij gegevens die ALFANUMERIEK gerepresenteerd worden is het mogelijk met de **wildcards** ’*****’ en/of ’**?**’ meer variaties aan te geven. Dit kan nuttig zijn als men een bepaalde selectie wil maken uit mogelijke waarden, of indien men onzeker is over de spellingswijze van bijvoorbeeld een naam.
Een ’*****’ in een alfanumerieke constante betekent dat op de plaats van de ’*’ **nul of meer** willekeurige karakters mogen staan. Bijvoorbeeld:

get werknemer **where** naam = "Jans*".

Namen als "Jansen", "Jansse", "Jans" en "Janssens" voldoen aan de gestelde voorwaarde.

Een ’**?**’ in een alfanumerieke waarde betekent dat op de plaats van de ’?’ **precies één** willekeurig karakter mag staan. Bijvoorbeeld:

get werknemer **where** naam = "Me?er".

Namen als "Meyer" en "Meier" voldoen aan deze voorwaarde.

Zowel "*" als "?" mogen op elke plaats binnen een alfanumerieke constante staan (vooraan, ergens tussenin of achteraan) en meerdere keren eventueel in combinatie voorkomen. Wildcards zijn overigens alleen mogelijk als de vergelijking een ’=’ (is gelijk aan) of ’<>’ (is ongelijk aan) betreft. Bij andere vergelijkingen (groter dan, kleiner dan etc.) worden de tekens ’*’ en ’?’ als zichzelf geïnterpreteerd.

III.3.4. Welke voorwaarden zijn mogelijk?

Eerst wordt aangegeven welke mogelijkheden het selecteren met behulp van één voorwaarde biedt. In het eerdere voorbeeld hebben we geselecteerd met behulp van de voorwaarde:

naam = "Pieterse"

Voor instances die aan de voorwaarde ’naam = "Pieterse"’ voldoen is de voorwaarde waar. In het Engels wordt dit aangegeven met behulp van de logische waarde **TRUE**.

Anderzijds geldt voor instances die niet aan de voorwaarde voldoen dat voor die instances de voorwaarde onwaar is. In het Engels **FALSE**. Aan een bepaalde voorwaarde wordt altijd **of** wel **of** niet voldaan. Dus altijd **of** TRUE **of** FALSE (tweewaardige logica).

Alleen instances waarvoor de voorwaarde waar is, worden geselecteerd.

In de voorwaarde ’naam = "Pieterse"’ wordt gebruik gemaakt van de relationele operator ’=’ (het ’is gelijk’-teken). Dit is echter niet de enige mogelijkheid. Hieronder staan alle toegestane relationele operatoren met betekenis opgesomd:

=	: is gelijk aan,
<>	: is ongelijk aan,
<	: is kleiner dan,
<=	: is kleiner dan of gelijk aan,
>	: is groter dan,
>=	: is groter dan of gelijk aan.

Een voorwaarde waarin gebruik gemaakt wordt van een relationele operator wordt een **vergelijking** genoemd.

Met deze operatoren kan men voorwaarden met de volgende combinaties van attributen en/of constanten opstellen:

- **constante met constante**.
  Voorbeelden hiervan zijn:

  2 = 3

Haal alle instances waarvoor geldt dat 2 gelijk aan 3 is. In dit geval is de voorwaarde FALSE en worden er dus geen instances gehaald.

2.5 <= 9

Haal alle instances waarvoor geldt dat 2.5 kleiner dan of gelijk is aan 9. De voorwaarden is TRUE dus worden alle instances gehaald.

"Bulletje" > "Bonestaak"

Haal alle instances waarvoor geldt dat Bulletje alfabetisch/lexicografisch groter is dan Bonestaak. Omdat Bulletje later in het alfabet voorkomt dan Bonestaak blijkt dat eerstgenoemde inderdaad groter is. Alle instances van het betreffende samengestelde type worden dus gehaald.

- **attribuut met constante.**
  Voorbeelden hiervan zijn:

  prijs > 1000

  Haal alle instances waarvoor de prijs groter is dan 1000. Men kan hierbij denken aan een get-opdracht op het type artikel of inkart.

  woonplaats <> "*dam"

  Haal alle instances waarvan de woonplaats niet eindigt op ’dam’. Men kan hierbij denken aan een get-opdracht op het type werknemer of klant.

- **attribuut met attribuut.**
  Voorbeelden hiervan zijn:

  aanbet > bedrag

  Haal de instances waarvoor geldt dat de aanbetaling groter is dan het bedrag van de verkoop. Hierbij moet men denken aan een get-opdracht op het type verkoop. Waarschijnlijk is de directeur van plan deze weldoeners hoogst persoonlijk een bedankbriefje te sturen.

  naam = afdeling its chef_naam

  Men moet hierbij denken aan een get-opdracht op het type werknemer. Haal de instances waarvoor geldt dat de naam van de werknemer gelijk is aan de naam van de chef van de afdeling waarop de werknemer werkt.

Gesteld dat in de gegeven voorbeelden alle mogelijke combinaties van domeinen van constanten en attributen binnen een vergelijking voorkomen, kan men de volgende conclusies trekken:

- ten aanzien van de domeinen REAL en INTEGER:
  - deze kunnen in dezelfde voorwaarde willekeurig naast elkaar gebruikt worden.
  - deze kunnen nooit vergeleken worden met attributen of constanten met domein ALFANUMERIEK en BOOLEAN.

- ten aanzien van domein ALFANUMERIEK:
  - deze kan alleen met attributen of constanten met domein ALFANUMERIEK vergeleken worden.
  - in constanten zullen '*' en '?' beschouwd worden als wildcards mits de relationele operator een '=' of '<>' is.

- ten aanzien van domein BOOLEAN:
  - deze kan nooit met behulp van een relationele operator, met een attribuut of constante van welk domein dan ook vergeleken worden. Dus ook niet met attributen of constanten met domein BOOLEAN. Dit kan een beperking lijken, maar is het niet. Immers, een voorwaarde die bestaat uit een vergelijking, en waarin dus geen BOOLEAN attributen en/of constanten gebruikt worden, geeft altijd een resultaat waarvan de waarde TRUE of FALSE is, dat wil zeggen: een resultaat met domein BOOLEAN. Het komt er dus op neer dat de resultaten van vergelijkingen en BOOLEAN constanten en/of expressies equivalent zijn.

Een bepaalde voorwaarde kan dus uit ofwel een vergelijking, ofwel een BOOLEAN attribuut of constante bestaan. Het resultaat van **iedere** voorwaarde heeft **altijd** domein BOOLEAN. Later zullen nog hulpmiddelen geïntroduceerd worden, waarmee het mogelijk wordt een aantal logische waarden te combineren tot een logische expressie.

Voorbeelden van **voorwaarden met een BOOLEAN** constante of attribuut zijn:

- **met een constante:**

  FALSE
  Bijv: **get** ... **where FALSE** (is equivalent met **get** ... **where FALSE = TRUE**).

  Haal alle instances waarvoor deze voorwaarde TRUE is.
  Deze voorwaarde is echter altijd FALSE zodat er *geen* instances gehaald worden.

  TRUE
  Bijv: **get** ... **where TRUE** (is equivalent met **get** ... **where TRUE = TRUE**).

  Haal alle instances waarvoor deze voorwaarde TRUE is.

Deze voorwaarde is altijd TRUE zodat *alle* instances gehaald worden.

Uit deze voorbeelden kan geconcludeerd worden dat het weinig zin heeft een voorwaarde met een BOOLEAN constante in de instance selectie specificatie op te nemen.

- **met een attribuut:**

  We nemen aan dat er in het conceptuele model een attribuut ’betaald’ voorkomt met een BOOLEAN domein.

  Bijv: **get** ... **where** betaald (is equivalent met: **get** ... **where** betaald = **TRUE**).

  Haal alle instances waarvoor de waarde van betaald TRUE is. Ogenschijnlijk een nogal triviale voorwaarde, maar desondanks een soort voorwaarde die veel gebruikt wordt.

In dit laatste voorbeeld blijkt dat het met de tot dusver behandelde hulpmiddelen bij selectie op BOOLEAN attributen alleen mogelijk is instances te selecteren waarvan het betreffende attribuut de waarde TRUE heeft. Dit is een beperking die vrij eenvoudig te verhelpen valt door de operator **not** in te voeren. De not-operator kan worden toegepast om de waarde van een voorwaarde te negateren. Hierbij geldt:

not TRUE is FALSE.
not FALSE is TRUE.

Enkele voorbeelden van voorwaarden waarin de not-operator gebruikt wordt:

not TRUE

Haal alle instances waarvoor de voorwaarde ’not TRUE’ waar is. Per definitie geldt dat de hier gespecificeerde voorwaarde altijd FALSE is. Er worden dus geen instances geselecteerd.

not betaald

Haal alle instances waarvoor de waarde van not betaald TRUE is.
Er geldt dat ’not betaald’ waar is indien betaald FALSE is. Immers, het negateren van FALSE levert TRUE op, en dus worden alle instances geselecteerd die de waarde FALSE voor attribuut betaald hebben.

not "Bulletje" > "Bonestaak"

Haal alle instances waarvoor Bulletje niet groter dan Bonestaak is. Dit wil dus zeggen: haal alle instances waarvoor Bulletje kleiner dan of gelijk aan Bonestaak is.

De volgende voorwaarde selecteert dus precies dezelfde verzameling instances:

"Bulletje" <= "Bonestaak"

Omdat Bulletje later in het alfabet komt dan Bonestaak is deze voorwaarde voor alle instances FALSE. Er worden dus geen instances geselecteerd.

De not-operator kan dus nuttig zijn bij BOOLEAN attributen. Bij BOOLEAN constanten is dit, evenals zonder not-operator, ook nu niet bijzonder zinvol.
Het blijkt tevens dat iedere vergelijking met not-operator ook zonder not geformuleerd kan worden. Immers:

=	en	<>	zijn complementair,
<	en	>=	zijn complementair,
>	en	<=	zijn complementair.

Er kunnen evenwel situaties zijn waarin het toch handig is bij vergelijking met de not-operator te werken. Bijvoorbeeld indien men met een bepaalde vergelijking juist een bepaalde verzameling instances heeft opgevraagd en men nu geïnteresseerd is in de complementaire verzameling. In dit geval is de tweede get-opdracht eenvoudig te formuleren door de not-operator aan de vergelijking toe te voegen.

III.3.5. Voorbeelden

Stel dat men alle werknemers wenst die op afdeling ’P&O’ werken. Hierbij is "P&O" de identificatie van de afdeling. Deze opvraging verloopt als volgt:

get werknemer **where** afdeling = "P&O".

Haal alle gegevens van de werknemers die op afdeling P&O werken.

Net als bij de selectie van attributen kan men bij de selectie van instances gebruik maken van de attributen van onderliggende samengestelde typen. Ook hier gebeurt dat door herhaald gebruik te maken van **its**.

Indien men weet dat met bovengenoemde identificatie de afdeling Peroneel en Organisatie bedoeld wordt, kan deze opvraging ook als volgt geformuleerd worden:

get werknemer **where** afdeling **its** afdnaam = "Peroneel en Organisatie".

Haal alle gegevens van de werknemers die op ’Peroneel en Organisatie’ werken.

Ook hier kan men, afhankelijk van het conceptuele model, doorgaan met het gebruik van its. Bij ieder volgend gebruik van its kan men steeds de attributen van het laatst gespecificeerde samengestelde type adresseren.

De opvraging van de verkopen van artikelen van het soort speelgoed verloopt als volgt:

get verkoop
where verkart **its** artikel **its** soort **its** beschrijving = "speelgoed".

Haal de gegevens van alle verkopen waarvan de artikelen van de soort speelgoed zijn.

Het blijkt dat ook hier alle attributen die in de ontledingsboom van het access-type liggen gespecificeerd kunnen worden. Het pad dat hierbij gevolgd wordt heet wederom het attribuutpad.

Natuurlijk is de combinatie van instance selectie en attribuut selectie ook mogelijk. Bijvoorbeeld bij de vraag van welke artikelen de voorraad kleiner is dan 10.

get artikel **its** beschrijving, voorraad
where voorraad < 10.

Haal de beschrijving en de voorraad van alle artikelen waarvan de voorraad kleiner is dan 10.

Overigens is onbelangrijk of alle woorden achter elkaar staan op één regel of verdeeld over meerdere regels. De afsluitende punt van de get-opdracht geeft het einde van de opdracht aan (zoals bij alle manipulatie-opdrachten). Vanwege de leesbaarheid wordt voor attribuut selectie (de lijst van attributen na de eerste its) en instance selectie (inclusief het woord ’where’) vaak ingesprongen.

III.3.6. Logische expressies

In het voorafgaande is een aantal voorbeelden gegeven van het samenstellen van voorwaarden zoals die binnen de instance selectie specificatie gebruikt kunnen worden. Deze voorwaarden resulteren bij uitvoering van de opdracht waarin ze gebruikt worden **altijd** in een logische waarde, dat wil zeggen **of** TRUE **of** FALSE.

Bij de get-opdracht met instance selectie zijn tot nu toe voorbeelden gegeven met slechts **één** voorwaarde. Hierbij was het zo dat alleen de instances waarvoor die voorwaarde waar was werden geselecteerd.

Dat dit lang niet altijd voldoende is werd duidelijk in het voorbeeld betreffende de heer Pieterse. Daar werd, naast naam, ook op voorletters geselecteerd.

Om deze mogelijkheid te bieden zijn de volgende twee logische operatoren gedefinieerd:

- de **and**-operator.
  Deze binaire operator heeft de volgende waarheidstabel:

FALSE **and** FALSE geeft FALSE.
FALSE **and** TRUE geeft FALSE.
TRUE **and** FALSE geeft FALSE.
TRUE **and** TRUE geeft TRUE.

De and-operator geeft dus alleen TRUE indien beide voorwaarden waarop deze wordt toegepast TRUE zijn. Indien **één** van de voorwaarden FALSE is, is het resultaat FALSE.

- de **or**-operator.
  Deze binaire operator heeft de volgende waarheidstabel:
  FALSE **or** FALSE geeft FALSE.
  FALSE **or** TRUE geeft TRUE.
  TRUE **or** FALSE geeft TRUE.
  TRUE **or** TRUE geeft TRUE.

  De or-operator geeft dus TRUE indien **één** van de voorwaarden waarop deze wordt toegepast TRUE is. Alleen als beide voorwaarden FALSE zijn is het resultaat FALSE.

Met behulp van deze operatoren en de al eerder genoemde not-operator is het mogelijk zogenaamde logische expressies samen te stellen. De prioriteit van de mogelijke operatoren hierin is van boven naar beneden:
- not,
- and,
- or.

Deze prioriteit geeft aan dat bij evaluatie van een bepaalde logische expressie eerst de not-operatie(s) op de betreffende voorwaarde uitgevoerd wordt, vervolgens de and-operatie(s) en als laatste de or-operatie(s). Met behulp van haken ’(’ en ’)’ kan de volgorde van afhandelen gewijzigd worden. De operator werkt dan op het resultaat van het gedeelte van de logische expressie dat tussen haken staat in plaats van slechts op de voorwaarde die direct links of rechts naast de operator staat. Merk overigens op dat de not-operator steeds op **één** logische waarde werkt terwijl de andere logische operatoren steeds op **twee** waarden werken. Zo’n logische waarde kan het resultaat zijn van **één** voorwaarde, dan wel logische expressie die omsloten is door haken. Enkele voorbeelden van logische expressies:

TRUE **and not** FALSE.

Eerst wordt de not-operator toegepast op FALSE, dit geeft TRUE. Vervolgens wordt de and-operator toegepast, op dit moment wordt dus de waarde bepaald van: TRUE **and** TRUE. Deze logische expressie geeft dus de waarde TRUE.

FALSE **or not** FALSE **and** TRUE.

Ook hier wordt eerst de not-operator uitgevoerd: **not** FALSE, dit geeft TRUE. Vervolgens de

and-operator: TRUE **and** TRUE, dit geeft TRUE. Als laatste wordt de or-operator uitgevoerd: FALSE **or** TRUE, dit geeft TRUE. Het resultaat van deze logische expressie is dus TRUE. Dit is natuurlijk een vrij verwarrend voorbeeld. Met behulp van haken zal deze expressie een stuk overzichtelijker worden:

FALSE **or** (**not** FALSE **and** TRUE), en
FALSE **or** ((**not** FALSE) **and** TRUE).

Deze logische expressies hebben hetzelfde resultaat maar zijn makkelijker te lezen.
De volgende logische expressies zijn echter qua uitwerkingsvolgorde niet hetzelfde:

(FALSE **or not** FALSE) **and** TRUE, en
FALSE **or not** (FALSE **and** TRUE).

Met behulp van deze logische expressies wordt het mogelijk op meerdere voorwaarden instances te selecteren door, in plaats van een enkele voorwaarde, logische expressies te accepteren in de instance selectie specificatie.

In het voorbeeld betreffende Pieterse wordt de and-operator gebruikt om naast op de naam ook op de voorletters te selecteren:

get werknemer **where** naam = "Pieterse" **and** voorl = "A.C.M.".

Haal alle gegevens van alle werknemers met naam ’Pieterse’ en voorletters ’A.C.M.’.

In deze opdracht wordt **’and’** gebruikt om aan te geven dat een bepaalde instance pas geselecteerd mag worden indien zijn naam ’Pieterse’ is en zijn voorletters ’A.C.M.’ zijn. Dus niet alle instances die in de vorige opvraging geselecteerd werden worden nu geselecteerd, omdat ze naast de gewenste naam niet de gewenste voorletters hebben. Dit laatste natuurlijk met uitzondering van de door ons gewenste ’Pieterse’, te weten ’A.C.M. Pieterse’.

Later volgen nog meer voorbeelden waarin van logische expressies gebruik gemaakt wordt om meerdere voorwaarden bij instance selectie te specificeren. Instance selectie is overigens de enige toepassing van logische expressies. Men kan dus geen logische expressies bij attribuut selectie toepassen. De volgende get-opdracht is dan ook **niet correct**:

get klant **its** schuld > 100.

III.3.7. Rekenkundige expressies

Tot nu toe is, zowel bij het selecteren van in het resultaat te plaatsen attributen als bij het selecteren van in het resultaat te verwerken instances, steeds gewerkt met waarden van attributen in precies dezelfde vorm als waarin deze in de database opgeslagen zijn. In de inleidende paragraaf van dit hoofdstuk is een tweetal voorbeelden gegeven waaruit blijkt dat

het soms wenselijk is ook met afbeeldingen van attributen te werken.

Deze afbeeldingen van attributen zijn alleen mogelijk met attributen die domein REAL of INTEGER hebben. Via rekenkundige expressies wordt de mogelijkheid hiertoe geboden. Hierbij zijn vier rekenkundige operatoren voorhanden. Deze zijn:

+	: optellen,
-	: aftrekken,
*	: vermenigvuldigen,
/	: delen.
%	: modulus

Vermenigvuldigen, delen en modulus hebben gelijke hoogste prioriteit, optellen en aftrekken gelijke laagste. Evenals bij logische expressies kan met behulp van de haken '(' en ')' de volgorde van afhandelen gewijzigd worden.

Hier wordt er vanuit gegaan dat men zodanig vertrouwd is met dit soort expressies dat voorbeelden hiervan direct met behulp van opdrachten kunnen worden gegeven.

Rekenkundige expressies kunnen zowel bij attribuut als bij instance selectie gebruikt worden.

III.3.7.1. Rekenkundige expressies bij attribuutselectie

Indien men bijvoorbeeld het in ieder artikel geïnvesteerde vermogen wil hebben dan wordt dit als volgt geformuleerd:

get artikel **its** voorraad * prijs.

Dit is een voorbeeld van een opvraging waarbij op slechts **één** samengesteld type wordt gewerkt. Hierin wordt met behulp van de rekenkundige expressie 'voorraad * prijs' een afbeelding gedefinieerd van attributen voorraad en prijs van het samengestelde artikel. Men kan echter ook bij het gebruik van expressies attributen overal uit de ontledingsboom van het access-type specificeren. Men kan bijvoorbeeld als volgt opvragen met hoeveel procent korting bepaalde artikelen in een bepaalde verkoop zijn verkocht:

get verkoop **its** (hoeveelheid * verkart **its** artikel **its** prijs - bedrag)
/ verkart **its** artikel **its** prijs * 100.

Natuurlijk kunnen naast afbeeldingen van attributen nog steeds de gebruikelijke opvragingen gedaan worden. Immers, welbeschouwd is de opvraging van attributen als zodanig ook een simpele afbeelding. Daarom zal tussen deze twee min of meer verschillende vormen van attribuutselectie in het vervolg geen onderscheid gemaakt worden. Steeds wordt onder attribuutselectie verstaan:

- attribuutselectie als zodanig, maar ook
- afbeeldingen van attributen m.b.v. rekenkundige expressies.

Nog een laatste voorbeeld van attribuutselectie waarin beide vormen willekeurig gebruikt worden:

```
get verkoop its
        klant its naam,
        verkart its artikel its beschrijving,
        (hoeveelheid * verkart its artikel its prijs - bedrag)
                / verkart its artikel its prijs * 100.
```

Hierbij wordt dus naast de verstrekte korting ook nog de naam van de klant en de beschrijving van het verkochte artikel gegeven.

III.3.7.2. Rekenkundige expressies bij instance-selectie

Men wil nu bijvoorbeeld een overzicht hebben van alle artikelen van de soort speelgoed waarvoor het geïnvesteerde vermogen groter is dan fl. 1.000,-.

```
get artikel its beschrijving, voorraad * prijs
        where soort its beschrijving = "speelgoed"
        and voorraad * prijs > 1000.
```

Haal van alle artikelen van de soort speelgoed waarvoor het geïnvesteerde vermogen groter is dan fl. 1.000,-, de beschrijving van het artikel en het hierin geïnvesteerde vermogen.

Ook bij instance-selecties kunnen rekenkundige expressies vrij gebruikt worden. Net als bij attribuutselectie zal daarom niet steeds specifiek vermeld worden dat dit het geval is, maar zal gesproken worden van gewoonweg instance-selectie. Bij instance-selectie kan dus gebruik gemaakt worden van:

- Logische expressies.
  De logische operatoren binnen zo'n logische expressie werken op ofwel een voorwaarde, ofwel het logische resultaat van het gedeelte van deze logische expressie dat door haken wordt omsloten en direct links of rechts van de betreffende operator staat.

- Voorwaarden.
  Deze kunnen bestaan uit:
  - een logische constante of attribuut, ofwel
  - het resultaat van een vergelijking. Binnen een vergelijking kan gebruik gemaakt worden van attributen en constanten met domein INTEGER, REAL en/of

ALFANUMERIEK. Alleen domeinen INTEGER en REAL kunnen in rekenkundige expressies gebruikt worden. Het domein ALFANUMERIEK mag als zodanig of in combinatie met een stringfunctie (bijvoorbeeld: combine, head, tail) gebruikt worden. Het type DATE mag alleen in combinatie met een datumfunctie (bijvoorbeeld: newdate, timedif, isdate, yearf, monthf, dayf, wdayf) gebruikt worden.

III.3.8 Wiskundige functies

Onderstaande tabel bevat alle ondersteunde wiskundige operatoren en functies die in rekenkundige expressies kunnen worden gebruikt. Rekenkundige expressies kunnen zowel bij attribuut - als bij instance selectie gebruikt worden.

operatornaam, zoals te gebruiken in Xplain	**rekenkundige functie**
+	optellen
-	aftrekken
*	vermenigvuldigen
/	delen
%	modulus
pow	machtsverheffen
abs	absolute waarde
max	maximale waarde
min	minimale waarde
sqrt	worteltrekken
exp	exponent-functie
ln	natuurlijke logaritme
log10	10-de logaritme
sin	sinus
cos	cosinus
tan	tangens
asin	arcsinus
acos	arccos

atan	arctangens
sinh	sinushyperbolicus
cosh	cosinushyperbolicus
tanh	tangenshyperbolicus
asinh	arcsinushyperbolicus
acosh	arccosinushyperbolicus
atanh	arctangenshyperbolicus

Gebruik van attributen in rekenkundige expressies:

Voorbeeld:

Stel een type A is als volgt gedefinieerd:

type A = B, C, D., waarbij B een getal is.

Indien men geïnteresseerd is in bijvoorbeeld de wortel van B dan kan dat binnen Xplain o.a. als volgt:

> **extend** A **with** wortel = 0.0.
> **update** A **its** wortel = **sqrt** (A **its** B) **where** A **its** B >= 0.
> **get** A **its** wortel.
>
> Breid het type A uit met een tijdelijk attribuut genaamd "wortel". Vervolgens wordt de wortel getrokken van attribuut B en opgeslagen in het uitgebreide attribuut wortel. Met de get-opdracht wordt tenslotte het resultaat opgevraagd.

Indien bijvoorbeeld attribuut C (van type A) een domein ALFANUMERIEK heeft en men probeert de wortel te trekken van attribuut C dan wordt de volgende foutmelding gegeven:

> type conflict: unexpected type
>
> syntax error in line <linenumber>: <syntax error>
> De verwerking van de query is afgebroken.

Enkele voorbeelden van rekenkundige functies m.b.v. value-opdrachten:

Machtsfunctie

Syntax machtsfunctie: pow (x1,x2).

Gebied van de machtsfunctie: 0 >= x1 >= 0 en 0 >= x2 >= 0.

pow-operator
Indien men 2 tot de macht 3 wilt uitrekenen dan kan dat binnen Xplain m.b.v. een value-opdracht als volgt:

> **value** macht = **pow**(2,3).
>
> Bepaal 2 tot de macht 3 en sla de waarde op in de variabele met de naam macht.
>
> **value** macht.
>
> Toon de waarde van variabele ’macht’.

Het resultaat van bovenstaande value-opdracht is dus een variabele met domein INTEGER. Deze variabele heeft de naam ’macht’. De waarde van de variabele is 8. Indien men 2.0 in plaats van 2 als parameter had opgegeven dan was de waarde van de variabele macht gelijk aan 8.0 (REAL).

Indien men een ander domein dan INTEGER OF REAL opgeeft dan wordt een fout melding gegeven.

Voorbeeld:

Stel dat men i.p.v. de parameters 2 en 3 de parameters 2 en p had opgegeven dan wordt de query afgebroken tenzij p een waarde is gegeven.

De foutmelding luidt als volgt:

> Identifier: ’p’ niet gedefinieerd.
> De verwerking van de query is afgebroken.

Absoluut-functie

Syntax absoluutfunctie: abs (x).
Gebied van de absoluutfunctie: 0 >= x >= 0.

abs-operator
Indien men de absolute waarde wil van een bepaalde berekening dan kan dat binnen Xplain als volgt:

> **value** absoluut = **abs**(**pow**(-2.0,3.0)).
> **value** absoluut.

Bepaal eerst -2.0 tot de macht 3.0 en sla de absolute waarde van de berekening in de variabele met de naam ’absoluut’. Toon daarna de waarde van de variabele absoluut.

Het resultaat van -2.0 tot de macht 3.0 is gelijk aan -8.0. Door het gebruik van de abs-operator wordt de absolute waarde, namelijk 8.0, in de variabele absoluut opgeslagen.

Maximum-functie

Syntax maximumfunctie: max (x1, x2).
Toepasbare domeinen: integer - , real - , alfanumeriek - en datum expressies.

max-operator
Indien men het maximum van een tweetal waarden wil kan dat als volgt:

```
value maximum = max (98, 12).
value maximum.
```

Het resultaat van deze functie is gelijk aan 98.

Minimum-functie

Syntax minimumfunctie: min (x1, x2).
Toepasbare domeinen: integer - , real - , alfanumeriek - en datum expressies.

min-operator
Indien men het minimum van een tweetal waarden wil kan dat als volgt:

```
value minimum = min ("aap", "beer").
value minimum.
```

Het resultaat van deze functie is gelijk aan "aap".

Wortel-functie

Syntax wortelfunctie: sqrt (x).
Gebied van de wortelfunctie: x >= 0.

sqrt-operator
Indien men bijvoorbeeld de wortel van de expressie 4 * 4 (=16) wilt uitrekenen dan kan dat binnen Xplain als volgt:

```
value wortel = sqrt(4 * 4).
value wortel.
```

Trek de wortel van de rekenkundige expressie ’4 * 4’ en bewaar de waarde in de variabele met de naam ’wortel’. Toon daarna de waarde van de variabele wortel.

Exponentiële functie

Syntax exponentiële functie: exp (x).
Gebied van de exponentiële functie: 0 >= x >= 0.

exp-operator
Indien men bijvoorbeeld e tot de macht 5 wilt uitrekenen dan kan dat binnen Xplain als volgt:

value exponent = **exp**(5).
value exponent.

Bepaal e tot de macht 5 en sla de berekende waarde in de variabele met de naam ’exponent’. Toon daarna de waarde van de variabele exponent.

Logaritmische functies (ln, log10)

Syntax logaritmische functies: ln (x) en log10 (x).
Gebied van de logaritmische functies: x > 0.

ln-operator
Dit is de natuurlijke logaritme. Indien men geïnteresseerd is in een de logaritme van een getal, bijvoorbeeld 7, dan kan dat binnen Xplain als volgt worden berekend:

value logaritme = **ln**(7).
value logaritme.

Bepaal de logaritme van het getal 7 en sla de berekende waarde in de variabele met de naam ’logaritme’. Toon daarna de waarde van de variabele logaritme.

log10-operator
Indien men de 10log van 1000 wilt uitrekenen dan kan dat binnen Xplain als volgt:

value logaritme = **log10**(1000.0).
value logaritme.

Bereken de 10logaritme van 1000.0 en sla de waarde op in de variabele met de naam ’logaritme’. Toon daarna de waarde van variabele logaritme.

Het resultaat van bovenstaande value-opdracht is dus een variabele met domein REAL, omdat als parameter een real wordt opgegeven.

Goniometrische functies (sinus, cosinus, tangens)

Syntax goniometrische functies: sin (x), cos (x) en tan (x).
Gebied van de goniometrische functies: 0 >= x >= 0.

Enkele voorbeelden:

sin-operator
Indien men geïnteresseerd is in bijvoorbeeld de sinus van 1 dan kan dat binnen Xplain als volgt:

> **value** sinus = **sin**(1).
> **value** sinus.
>
> Bepaal de sinus van het gehele getal 1 en sla de waarde op in de variabele met de naam 'sinus'. De tweede opdracht toont de waarde van de variabele sinus.

cos-operator
Indien men geïnteresseerd is in bijvoorbeeld de cosinus van 1 dan kan dat binnen Xplain als volgt:

> **value** cosinus = **cos**(1).
> **value** cosinus.
>
> Bepaal de cosinus van het gehele getal 1 en sla de waarde op in de variabele met de naam 'cosinus'. Toon de waarde van de variabele cosinus.

tan-operator
En, indien men geïnteresseerd is in bijvoorbeeld de tangens van 1 dan kan dat binnen Xplain als volgt:

> **value** tangens = **tan**(1).
> **value** tangens.
>
> Bepaal de tangens van het gehele getal 1 en sla de waarde op in de variabele met de naam 'tangens'. De tweede opdracht toont de waarde van de variabele tangens.

Inverse goniometrische functies (arcsinus, arccosinus, arctangens)

Syntax inverse goniometrische functies: asin (x), acos (x) en atan (x).
Gebied van de inverse functies arcsinus en arccosinus: - 1 < x < 1.
Gebied van de inverse functie arctangens: [- 'oneindig' < x < + 'oneindig'].

Enkele voorbeelden:

asin-operator
Indien men geïnteresseerd is in bijvoorbeeld de inverse sinus (arcsinus) van 0 dan kan dat binnen Xplain als volgt:

value arcsinus = **asin**(0).
value arcsinus.

Bepaal de arcsinus van het gehele getal 0 en sla de waarde op in de variabele met de naam 'arcsinus'. De tweede opdracht toont de waarde van de variabele arcsinus.

acos-operator
Indien men geïnteresseerd is in bijvoorbeeld de inverse cosinus (arccosinus) van 0 dan kan dat binnen Xplain als volgt:

value arccosinus = **acos**(0).
value arccosinus.

Bepaal de arccosinus van het gehele getal 0 en sla de waarde op in de variabele met de naam 'arccosinus'. De tweede opdracht toont de waarde van de variabele arccosinus.

atan-operator
Indien men geïnteresseerd is in bijvoorbeeld de inverse tangens (arctangens) van 1 dan kan dat binnen Xplain als volgt:

value arctangens = **atan**(1).
value arctangens.

Bepaal de arctangens van het gehele getal 1 en sla de waarde op in de variabele met de naam 'arctangens'. De tweede opdracht toont de waarde van de variabele arctangens.

Hyperbolische functies (sinh, cosh, tanh)

Deze functies zijn meetkundig met behulp van de orthogonale hyperbool x^2 - y^2 = 1 weer te geven.

Definitie sinus hyperbolicus:

sinh x = (e^x - e^x) / 2.

Definitie cosinus hyperbolicus:

cosh x = (e^x + e^x) / 2.

Definitie tangens hyperbolicus:

tanh x = (e^x - e^x) / (e^x + e^x).

Syntax hyperbolische functies: sinh (x), cosh (x) en tanh (x).
Gebied van de hyperbolische functies: 0 >= x >= 0.

Enkele voorbeelden:

sinh-operator
Indien men geïnteresseerd is in bijvoorbeeld de sinushyperbolicus van 0 dan kan dat binnen Xplain als volgt:

value sinhyp = **sinh**(0).
value sinhyp.

Bepaal de sinushyperbolicus van het gehele getal 0 en sla de waarde op in de variabele met de naam 'sinhyp'. De tweede opdracht toont de waarde van de variabele sinhyp.

cosh-operator
Indien men geïnteresseerd is in bijvoorbeeld de cosinushyperbolicus van 0 dan kan dat binnen Xplain als volgt:

value coshyp = **cosh**(0).
value coshyp.

Bepaal de cosinushyperbolicus van het gehele getal 0 en sla de waarde op in de variabele met de naam 'coshyp'. De tweede opdracht toont de waarde van de variabele coshyp.

tanh-operator
Indien men geïnteresseerd is in bijvoorbeeld de tangenshyperbolicus van 0 dan kan dat binnen Xplain als volgt:

value tanhyp = **tanh**(0).
value tanhyp.

Bepaal de tangenshyperbolicus van het gehele getal 0 en sla de waarde op in de variabele met de naam 'tanhyp'. De tweede opdracht toont de waarde van de variabele tanhyp.

Inverse hyperbolische functies (arcsinh, arccosh, arctanh)

Definitie en gebied arcsinus hyperbolicus:

arcsinh x = ln(x + sqrt(x^2 + 1)) [0 >= x >= 0]

Definitie en gebied arccosinus hyperbolicus:

arccosh x = ln(x + sqrt(x^2 -1)) [x >= 1]
of
arccosh x = ln(x - sqrt(x^2 -1)) [x >= 1]

Definitie en gebied arctangens hyperbolicus:

arctanh x = 0.5 * ln((x +1)/(x-1)) [-1 <= x <= +1]

Syntax inverse hyperbolische functies: asinh (x), acosh (x) en atanh (x).
Gebied van de inverse hyperbolische functies: zie hierboven.

Enkele voorbeelden:

asinh-operator
Indien men geïnteresseerd is in bijvoorbeeld de arcsinushyperbolicus van 5 dan kan dat binnen Xplain als volgt:

value asinhyp = **asinh**(5).
value asinhyp.

Bepaal de arcsinushyperbolicus van het gehele getal 5 en sla de waarde op in de variabele met de naam ’asinhyp’. De tweede opdracht toont de waarde van de variabele asinhyp.

acosh-operator
Indien men geïnteresseerd is in bijvoorbeeld de arccosinushyperbolicus van 5 dan kan dat binnen Xplain als volgt:

value acoshyp = **acosh**(5).
value acoshyp.

Bepaal de arccosinushyperbolicus van het gehele getal 5 en sla de waarde op in de variabele met de naam ’acoshyp’. De tweede opdracht toont de waarde van de variabele acoshyp.

atanh-operator
Indien men geïnteresseerd is in bijvoorbeeld de arctangenshyperbolicus van 5 dan kan dat binnen Xplain als volgt:

value atanhyp = **atanh**(5).
value atanhyp.

Bepaal de arctangenshyperbolicus van het gehele getal 5 en sla de waarde op in de variabele met de naam 'atanhyp'. De tweede opdracht toont de waarde van de variabele atanhyp.

III.3.9. Set-functies

Naast de get-opdracht met gebruik van afbeeldingen bij attribuut selectie bestaat er nog een familie van get-opdrachten waarbij de uit de database gehaalde gegevens niet als zodanig in de database staan. Het betreft hier de get-opdracht met gebruik van een set-functie. Deze familie van get-opdrachten komt voort uit de wens om nieuwe informatie uit reeds opgeslagen gegevens te kunnen afleiden.

Voorbeelden van het bedoelde soort opvragingen:

- Geef het aantal stuks dat van een bepaald artikel verkocht is.
- Geef de klant die in het afgelopen jaar voor het grootste bedrag artikelen heeft gekocht en betaald.
- Geef de leverancier die een bepaald artikel voor de laagste prijs verkoopt.
- Geef het totale vermogen dat in alle artikelen van de soort speelgoed is geïnvesteerd.

Om dit soort opvragingen te kunnen doen zijn de volgende set-functies beschikbaar:

- **max** levert de grootste waarde van een collectie van attribuutwaarden.
- **min** levert de kleinste waarde van een collectie van attribuutwaarden.
- **total** levert de som van de waarden in de collectie van attribuutwaarden.
- **count** levert het aantal elementen van een collectie.
- **any** levert de logische waarde TRUE indien de geselecteerde collectie *minstens één* instance bevat, anders FALSE.
- **nil** levert de logische waarde TRUE indien de geselecteerde collectie *geen* instances bevat, anders FALSE.
- **some** levert een willekeurig element uit de geselecteerde collectie.

De get-opdracht met set-functie is een opdracht waarin steeds eerst, m.b.v. attribuut-selectie en/of instance-selectie, **één** collectie uit de database gelicht wordt. Vervolgens wordt op deze collectie de betreffende set-functie toegepast.
Kenmerkend voor set-functies is dat ze als resultaat **altijd precies één** waarde geven. We kunnen de verschillende set-functies in 3 categorieën opdelen:

- Set-functies die op een collectie attribuutwaarden werken en op attribuutniveau bepaalde dingen over deze collectie afleiden. Dit zijn de set-functies: **max, min** en **total**. De eerste twee zijn in feite relationele set-functies en kunnen derhalve op attributen met

ieder domein behalve BOOLEAN worden toegepast. De laatste set-functie is een rekenkundige en kan derhalve alleen op attributen met domein REAL of INTEGER worden toegepast.

- Set-functies die op een collectie instances werken en op instance-niveau bepaalde dingen over deze collectie afleiden. Dit zijn de set-functies: **count, any** en **nil**. In feite zijn deze functies op het aantal instances in de geselecteerde collectie gebaseerd.

- Set-functies die op een collectie instances werken en die op zowel instance- als attribuutniveau iets met de instances in deze collectie kunnen doen. Hiermee wordt de set-functie **some** bedoeld. Deze set-functie haalt een willekeurige instance uit de geselecteerde collectie. Vervolgens kan net als met de get-opdracht zonder set-functie attribuut-selectie op het gespecificeerde type worden uitgevoerd.

Bij de eerstgenoemde categorie set-functies (max, min en total) **moet** er **precies één** attribuut of attribuut-afbeelding geselecteerd worden. Het domein van het attribuut of de afbeelding moet overeenkomen met de toegestane domeinen van de betreffende set-functie. Bij de tweede categorie set-functies (count, any en nil) **mag** er **hoogstens één** attribuut of attribuut-afbeelding geselecteerd worden. Het attribuut-domein is hierbij niet beperkt. Bij de derde categorie **mag** net als bij de tweede categorie **hoogstens één** attribuut geselecteerd worden. Bij alle set-functies kan natuurlijk ook instance-selectie plaatsvinden.

Overigens is het woord ’set’ in ’set-functie’ niets anders dan de Engelse vertaling van het Nederlandse woord ’verzameling’. Met nadruk zij vermeld dat hier **niet** het wiskundige begrip ’verzameling’ bedoeld wordt.

In het volgende wordt de get-opdracht met gebruik van een set-functie voor ieder van de verschillende functies nader bekeken.

Max

Deze set-functie selecteert het maximum uit een collectie van attribuutwaarden, bijvoorbeeld:

get max artikel **its** prijs.

Geef de prijs van het duurste artikel.

Maar ook:

get max werknemer **its** afdeling **its** afdnaam.

Geef de naam van de afdeling waarop iemand werkt en die als laatste afdnaam in het alfabet voorkomt.

Het blijkt dus dat men met behulp van de max-functie het maximum kan bepalen van een collectie van attributen van het typen REAL, INTEGER of ALFANUMERIEK. Hierbij is willekeurige attribuut-selectie en instance-selectie toepasbaar zolang maar wel steeds **één** attribuut wordt geselecteerd. Nog een voorbeeld:

get max artikel **its** prijs * voorraad
where soort **its** beschrijving = "speelgoed".

Geef het grootste vermogen dat in een artikel van de soort speelgoed is geïnvesteerd.

In dit voorbeeld wordt de set-functie dus toegepast op de door middel van ’prijs * voorraad’ verkregen collectie van attribuut-afbeeldingen.

Min

Deze set-functie selecteert het minimum uit een collectie van attribuutwaarden.
Het gebruik van de min-functie is analoog aan dat van de max-functie. Hier wordt volstaan met een voorbeeld: De waarde van de goedkoopste aankoop van een artikel:

get min inkart **its** prijs.

Total

Deze set-functie levert de som (het totaal) van een collectie van attribuutwaarden.
Deze attributen moeten dan wel domein REAL of INTEGER hebben. Het gebruik van deze set-functie is verder hetzelfde als dat van de **min** en **max** set-functies. Bijvoorbeeld:

get total artikel **its** prijs * voorraad
where soort **its** beschrijving = "speelgoed".

Geef het totaal geïnvesteerde vermogen in alle artikelen van de soort speelgoed.

Count

De count-functie telt het aantal instances van een bepaalde collectie van instances. De count-functie geeft altijd een INTEGER-waarde terug die groter of gelijk aan nul is.
Het aantal werknemers wordt bijvoorbeeld als volgt opgevraagd:

get count werknemer.

Geef het aantal werknemers.

Het aantal werknemers dat op de afdeling ’speelgoed’ werkt wordt als volgt opgevraagd:

get count werknemer
where afdeling **its** afdnaam = "speelgoed".

Zojuist is al opgemerkt dat attribuutselectie niet nodig is. Het is echter wel mogelijk, bijvoorbeeld:

get count verkoop **its** verkart.

Geef het aantal verschillende artikelen dat verkocht is.

en

get count verkoop **its** verkart **its** afdeling.

Geef het aantal verschillende afdelingen dat verkopen gedaan heeft.

De volgende opvraging zou het aantal maal dat er instances van klant en aanbet in verkoop voorkomen moeten geven:

get count verkoop **its** klant, aanbet.

Hier is echter een telling van **twee** collecties gespecificeerd. Dit is **geen correcte opdracht** omdat het resultaat van een set-functie **enkelvoudig** dient te zijn. De opvraging kan gesplitst worden in twee aparte opvragingen.

Een andere mogelijkheid is de functie **combine** (zie verder) te gebruiken. Aangezien aanbet niet van type integer/string is dient ook nog de conversie functie **string** te worden toegepast:

get count verkoop **its combine**(klant, **string**(aanbet)).

Any

De any-functie geeft altijd een BOOLEAN-waarde terug. De set-functie geeft de waarde TRUE indien er **minstens één** instance in de geselecteerde collectie zitten, anders FALSE. Het gebruik van deze set-functie is hetzelfde als bij de **count** set-functie. Dus bijvoorbeeld:

get any artikel **where** prijs > 200.

Geeft de waarde TRUE indien er een artikel bestaat met een prijs groter dan fl. 200,-.

Nil

Deze set-functie geeft de waarde TRUE indien er **geen** instances in de geselecteerde collectie zitten, anders FALSE. Deze set-functie is complementair aan de **any** set-functie, bijvoorbeeld:

get nil verkoop **where** aanbet > bedrag.

Geeft de waarde TRUE indien er geen verkopen zijn waarvoor de aanbetaling groter is dan het bedrag.

Indien het resultaat van deze opvraging FALSE is dan bestaat er minstens één instance van verkoop waarbij meer aanbetaald is dan nodig.

Some

Deze set-functie geeft als resultaat een willekeurige instance van het access-type dat aan de gespecificeerde voorwaarden voldoet. Attribuutselectie kan gebruikt worden op de gebruikelijke manier, bijvoorbeeld:

get some verkoop **its** bedrag / 1.175 **where yearf**(datum) > 1990.

Geef het bedrag exclusief 17,5% omzetbelasting van een willekeurige verkoop gedaan na 1990.

Er mag ook hier **maximaal één** attribuutselectie plaatsvinden. Indien er geen attribuut geselecteerd wordt, dan wordt als resultaat **alleen** de identificatie gegeven. Dit in tegenstelling tot attribuutselectie m.b.v. de get-opdracht zonder set-functie waar in dit geval **ook** nog **alle** attributen gegeven zouden worden.

Indien er **wel** een attribuut geselecteerd wordt, dan wordt **alleen** dit attribuut als resultaat gegeven, en dus niet ook nog de identificatie.

De set-functie ’some’ geeft in beide gevallen altijd precies **één** waarde als uitvoer, althans indien het aantal instances van het betreffende type niet nul is. Later, bij de behandeling van specialisatie zullen nog voorbeelden volgen waaruit het nut van ’some’ blijkt.

Dit voorbeeld geeft trouwens ook aan dat door opslag van de datum in de vorm jjjjmmdd, met jjjj voor jaarnummer, mm voor maandnummer en dd voor dagnummer, het mogelijk wordt allerlei zinvolle vergelijkingen betreffende een bepaalde datum of periode op te stellen.

III.3.10. String-functies

De volgende drie functies zijn beschikbaar voor stringmanipulatie:

- **combine**: levert als resultaat de aaneenschakeling van twee strings. Dit resultaat wordt als één waarde opgevat. De functie is alleen toepasbaar op waarden van het type integer en string.
- **head**: levert het eerste gedeelte van een string totdat de eerste spatie of het einde van de string bereikt wordt. De functie is alleen toepasbaar op waarden van het type string.

- **tail**: levert het restant nadat de eerste gedeelte uit de string verwijderd is. In geval het resultaat leeg is, wordt 1 spatie afgeleverd. De functie is alleen toepasbaar op waarden van het type string.

In alle gevallen is het resultaat van type string. Het domein wordt door Xplain bepaald. Alleen de stringfunctie **combine** mag worden toegepast op (attribuut/variable-) expressies van het type integer of string. De functies **head** en **tail** zijn daarentegen alleen zinvol (en daarom toegestaan) bij argumenten van het type string. Hieronder volgen enkele voorbeelden.

Combine

Stel de volgende typedefinitie is aanwezig:
type klant = naam, voorletters, adres, woonplaats.

identificatie klant	naam	voorl.	adres	woonplaats
123456	van Dalen	A H	Straatweg 5	Amsterdam
312645	Boogschutter	J	Peensteeg 7	Rotterdam
981265	Put	J M H	Keetweg 1	Amsterdam

De concatenatie (samenvoeging) van naam en voorletters resulterend in het attribuut persoonsnaam verkrijgt men bijv. door de volgende opdracht:

extend klant **with** persoonsnaam = **combine**(voorletters, **combine**(" ", naam)).

persoonsnaam
A H van Dalen
J Boogschutter
J M H Put

eerste
A
J
J

restant
H
-
M H

Zie ook functies **head** en **tail**.

N.B. - betekent 1 spatie !

Er geldt: domein(persoonsnaam) = domein(voorletters) + domein (" ") + domein(naam).
Aangezien de stringfuncties ook mogen worden toegepast op complexe stringexpressies is de volgende extensie ook mogelijk:

extend klant **with** complex =
combine ("De heer ", **combine** (voorletters, **combine**(" ", naam))).

Head

De functie **head** resulteert in het eerste veld binnen een stringattribuut. Een veld wordt hierbij (zoals gebruikelijk) gedefinieerd door een (eventueel lege) groep van te printen karakters afgesloten met een spatie. Het resultaat van de stringfunctie bestaat altijd uit tenminste uit een

enkele spatie. Het domein van het resultaat is gelijk aan het domein van het argument.

Voorbeeld:

extend klant **with** eerste = **head** (voorletters).

Resulteert in de eerste voorletter, dus in het onderhavige voorbeeld in: A, J, J (zie voorgaande tabel onder **combine**).

Tail

De functie **tail** resulteert in het restant van de string nadat de head hieruit verwijdert is. Het domein is gelijk aan het domein van het argument. Een voorbeeld ter illustratie (zie ook voorgaande tabel):

extend klant **with** restant = **tail** (voorletters).

III.3.11. Datumfuncties

De volgende functies maken operaties op waarden van het type datum mogelijk:
- **newdate**: Bepaalt een nieuwe kalenderdatum die op een bepaalde afstand van een gegeven kalenderdatum ligt (resultaat: date).
- **timedif**: Bepaalt het aantal dagen verschil tussen twee gegeven kalenderdatums (resultaat: integer).
- **isdate**: Bepaalt of een waarde een geldige kalenderdatum voorstelt (resultaat: Boolean TRUE of FALSE).
- **yearf**: Bepaalt het jaar van een gegeven kalenderdatum (resultaat: integer).
- **monthf**: Bepaalt de maand van een gegeven kalenderdatum (resultaat: integer met bereik 1..12).
- **dayf**: Bepaalt de maanddag van een gegeven kalenderdatum (resultaat: integer met bereik 1..31).
- **wdayf**: Bepaalt de weekdag van een gegeven kalenderdatum (resultaat: integer met bereik 0..6 waarbij zondag = 0, maandag = 1, dinsdag = 2, etc.).

Voorbeelden van datumfuncties:
- newdate (19970224, 10) resultaat: 19970306.
- timedif(19970401, 19970325) resultaat: 7.
- isdate (19970229) resultaat: FALSE.
- yearf(1999970228) resultaat: 1997.
- monthf(19970228) resultaat: 2.
- dayf(19970228) resultaat: 28.
- wdayf(19970228) resultaat: 5.

III.3.12. Conversiefuncties

De functies **integer**(a), **real**(a), **string**(a) en **datef**(a) maken het mogelijk om het domein van een integer- real-, string- of date-expressie te veranderen. Dit heeft ondermeer gevolgen voor de representatie van een resultaat. Het is bijvoorbeeld mogelijk om de integerwaarde

jaar * 10000 + maand * 100 + dag

op te vatten als datum, met de conversiefunctie datef, als volgt:

datef(jaar * 10000 + maand * 100 + dag)

Door conversie kan de weergave veranderen. Als men bijvoorbeeld een numerieke waarde converteert tot string, dan wijzigt hiermee ook de weergave van rechtsaansluitend in linksaansluitend.

III.3.13. Systeemvariabelen

Het Xplain DBMS kent twee systeemvariabelen die gebruiker zonder eigen declaratie kan gebruiken. Deze variabelen verschaffen de gebruiker informatie over de toestand waarin het systeem gebruikt wordt.

Variabele: systemdate

De variabele met de gereserveerde naam **systemdate** wordt door Xplain gegenereerd en bevat de datum waarop de huidige gebruiker heeft ingelogd bij Xplain. Deze variabele heeft domein DATE en heeft de interpretatie als hierboven in dee vorm jjjjmmdd. D.w.z. vier cijfers jjjj voor de jaaraanduiding, twee cijfers mm voor de maand aanduiding en twee cijfers dd voor dagaanduiding. Systemdate kan bijvoorbeeld als volgt met een value opdracht worden opgevraagd:

value datum = **systemdate**.

Op **systemdate** zijn de operaties uit te voeren door datumfuncties te gebruiken.

Variabele: loginname

De variabele met de gereserveerde naam **loginname** wordt door Xplain gegenereerd en bevat de naam van de gebruiker die het systeem momenteel gebruikt. Deze variabele heeft domein A13. Loginname kan bijvoorbeeld als volgt met een value opdracht worden opgevraagd:

value gebruiker = **loginname**.

Ook **loginname** mag bij database opdrachten worden gebruikt.

III.4. VALUE-OPDRACHT

III.4.1. Inleiding

Met behulp van de value-opdracht wordt de mogelijkheid geboden om het resultaat van

- een rekenkundige expressie,
- een logische expressie,
- een set-functie of
- een input-functie

in een **tijdelijke** variabele op te slaan. Zo'n variabele kan slechts **één** waarde bevatten. Het domein van de variabele kan REAL, INTEGER, ALFANUMERIEK of BOOLEAN zijn en wordt bepaald door de expressie of functie die de variabele een waarde geeft. De variabele is tijdelijk omdat deze na het beëindigen van de query automatisch verwijderd wordt. De value-opdracht is alleen bedoeld om in de query tussentijds waarden op te slaan en te gebruiken in vervolg-opdrachten. Constanten die permanent in de database opgeslagen moeten worden, kunnen gedefinieerd worden bij datadefinitie met de **constant**-opdracht.

De waarde, die met behulp van de value-opdracht afgeleid is en in de variabele geplaatst is, kan op twee manieren gebruikt worden:

1) als eindresultaat van een opvraging.
2) als tussenresultaat binnen een opvraging waarvoor het noodzakelijk is meerdere opdrachten uit te voeren. De betreffende waarde kan daarom in alle opdrachten, dus ook in de value-opdracht zelf, gebruikt worden.

Eerst worden de verschillende vormen van de value-opdracht behandeld. Hierbij moet onderscheid gemaakt worden tussen:

- value-opdrachten waarin **niet** gerefereerd wordt aan het conceptuele model.
  Er is sprake van een 'constante waarde'-toekenning, een berekening (expressie zonder attributen uit het model) of een vraag voor gebruikersinvoer (input-functie).

- value-opdrachten waarin **wel** gerefereerd wordt aan het conceptuele model.
  Er moet altijd eerst attribuut en instance selectie op een bepaald samengesteld type uitgevoerd worden. Omdat met behulp van de value-opdracht slechts **één** waarde kan worden opgeslagen moet afgedwongen worden dat zo'n selectie in niet meer dan **één** waarde resulteert. Omdat **set-functies** deze gewenste eigenschap hebben moeten deze in dit geval **altijd** gebruikt worden.

III.4.2. Zonder set-functie

In dit geval mag de waarde die men wil opslaan **nooit** het resultaat zijn van een bepaalde

selectie op de database. Er moet dus sprake zijn van een bepaalde berekening waarin niet gerefereerd wordt aan (basis-)typen in het conceptuele model.

Een eerste voorbeeld van dit soort value-opdracht is:

> **value** getal1 = 1.
>
> Bewaar het resultaat van de rekenkundige expressie ’1’ in de variabele met naam ’getal1’.

Men moet iedere variabele een naam geven die nog niet binnen het systeem gebruikt wordt. Dus, indien men op een gegeven moment een bepaalde value-opdracht wil uitvoeren, dan mag de naam van de variabele waarin men de waarde wil bewaren niet gelijk zijn aan de:
- namen van (basis-)typen of ’variable’n in het conceptuele model.
- namen van standaardwoorden zoals: **’get’, ’its’, ’and’, ’max’** en dergelijke.

Men mag een bepaalde variabele wel de naam geven van een reeds bestaande valuevariabele. De valuevariabele wordt dan geherdefinieerd en de oude waarde zal hierbij verloren gaan.

Namen van variabelen mogen bestaan uit maximaal 20 letters en/of cijfers en moeten beginnen met een letter. Er wordt onderscheid gemaakt tussen kleine en hoofdletters. Spaties en andere speciale tekens (zoals /, -, _, #, enz.) mogen niet gebruikt worden.

Het resultaat van bovenstaande value-opdracht is dus een variabele met domein INTEGER. Deze variabele heeft de naam ’getal1’. De waarde van deze variabele is 1. De variabele kan men nu gebruiken in expressies en selecties. Men kan de waarde van de variabele ook direct tonen. Dit is mogelijk met de volgende value-opdracht:

> **value** getal1.
>
> Toon de waarde van variabele ’getal1’.

Dit is dus een value-opdracht waarbij het gedeelte vanaf ’=’ ontbreekt. Dit gedeelte specificeert bij de value-opdracht namelijk altijd de waarde die de betreffende variabele moet krijgen. Dit wordt daarom de **value-specificatie** genoemd. Indien de value-specificatie ontbreekt wil men de betreffende variabele kennelijk geen nieuwe waarde geven. Men krijgt dan de waarde van de value te zien.

Nog enkele voorbeelden:

> **value** somvanbreuken = 2/5 + 3/7.
>
> Bewaar het resultaat van de rekenkundige expressie ’2/5 + 3/7’ in de variabele met de naam ’somvanbreuken’.

value gebrokengetal1 = 1.0.

Het domein van ’somvanbreuken’ en ’gebrokengetal1’ is REAL.

value waar = (TRUE).

Bewaar het resultaat van de logische expressie ’TRUE’ in de variabele met de naam ’waar’.

value isgroter = (2 > 0).

Bewaar het resultaat van vergelijking ’2 > 0’ in de variabele met de naam ’isgroter’. Deze vergelijking heeft TRUE als resultaat.

Het domein van ’waar’ en isgroter’ is BOOLEAN.

value zoeknaam = "Koolmees".

Het domein van ’zoeknaam’ is ALFANUMERIEK.

III.4.3. Met set-functie

In dit geval wil men een waarde hebben die op een bepaalde wijze uit de database bepaald moet worden. Het gebruik van set-functies bij een value-opdracht is precies hetzelfde als bij de get-opdracht (zie paragraaf III.3.9). Nu wordt alleen de waarde van de get-opdracht in een variabele opgeslagen. Een voorbeeld hiervan is:

value maxprijs = **max** artikel **its** prijs.

Bewaar de prijs van het duurste artikel in de variabele met de naam ’maxprijs’.

III.4.4. Met input-functie

Met de input-functie kan men de gebruiker vragen één waarde in te voeren. Deze waarde wordt dan opgeslagen in de variabele. De variabele kan dan in de opvolgende opdrachten gebruikt worden (o.a. bij selecties of berekeningen). Men kan zo queries interactief maken zodat ze algemener bruikbaar worden. Een voorbeeld hiervan is:

value datum = **input** (D).

Vraagt de gebruiker een datum die wordt opgeslagen in de tijdelijke variabele ’datum’.

Op scherm krijgt de gebruiker het volgende te zien:

datum = input(D) : .....

De cursor zal achter de tekst (in de .....) staan. Het programma zal nu wachten op invoer van de gebruiker. Met <Return> kan de gebruiker aangeven dat men klaar is met invoeren. Daarna zal de invoer gecontroleerd worden op geldigheid. Als de invoer niet voldoet aan het gespecificeerde domein (in het bovenstaande geval een DATE) dan wordt er een foutmelding gegeven en stopt de verwerking van de query, anders gaat de verwerking gewoon door.

Voor de ALFANUMERIEKE waarden moet men '(An)' gebruiken i.p.v. '(In)' waarbij voor n een positief getal gesubstitueerd moet worden. Voor BOOLEAN waarden moet men '(B)' gebruiken. Voor gebroken getallen moet men '(Rx,y)' gebruiken waarbij x het aantal cijfers voor de decimale punt aangeeft en y het aantal cijfers achter de decimale punt. Bijvoorbeeld:

value minimumprijs = **input** (R6,2).

De standaardtekst die voor het invoerveld getoond wordt zal niet voor iedereen duidelijk genoeg of gewenst zijn. Het is dan ook mogelijk om een andere begeleidende voorlooptekst mee te geven achter de input-functie, bijvoorbeeld:

value maxprijs = **input** (R6,2) "Wat is de maximum prijs? ".

geeft tijdens uitvoering het volgende te zien:

Wat is de maximum prijs? .....

Men kan tijdens verwerking ook gebruik maken van een defaultwaarde door de variable vooraf (d.w.z. voor de input-opdracht) reeds van een toegestane waarde te voorzien.

III.4.5. Gebruik van variabelen in opdrachten

Hoewel alleen de get- en value-opdracht aan de orde geweest zijn, geldt de hier getoonde wijze van gebruik van variabelen ook voor de nog te behandelen opdrachten.
Het gebruik van variabelen in opdrachten is vrij eenvoudig. Men kan variabelen overal gebruiken waar attributen en/of constanten gebruikt mogen worden. Dit geldt dus voor zowel attribuut- als instance-selectie binnen opdrachten betreffende welk samengesteld type dan ook, maar ook voor value-opdrachten zonder set-functie. Enkele voorbeelden:

value maxprijs = **max** artikel **its** prijs.
get artikel **where** prijs = maxprijs.

Haal alle gegevens van de artikelen die de hoogst voorkomende prijs hebben.

In dit voorbeeld wordt dus eerst m.b.v. de value-opdracht de prijs van het duurste artikel bepaald. Vervolgens worden m.b.v. de get-opdracht die instances van artikel geselecteerd

waarvoor de prijs gelijk is aan de prijs van het duurste artikel. Het voordeel van het gebruik van variabelen blijkt hier uit het feit dat men naast de hoogste prijs waarvoor een artikel verkocht wordt ook de artikelen kan bepalen die voor deze hoogste prijs verkocht worden.

Het bepalen van de op één na duurste artikelen gaat bijvoorbeeld als volgt:

```
value maxprijs =
        max artikel its prijs.
value opeennamaxprijs =
        max artikel its prijs
        where prijs < maxprijs.
get artikel where prijs = opeennamaxprijs.
```

De mogelijkheden van het gebruik van variabelen in get-opdrachten worden nog eens geïllustreerd in het volgende voorbeeld:

```
get verkart its maxprijs / 100.
```

Haal voor ieder verkocht artikel; maxprijs / 100.

Het resultaat van deze opvraging is een tabel met voor iedere instance van verkart de betreffende identificatie en steeds dezelfde waarde van maxprijs / 100.
Dit is een vrij zinloze opvraging, maar illustreert dat:

- variabelen net als attributen en constanten in expressies gebruikt kunnen worden. De voorwaarden voor gebruik zijn hierbij gelijk aan die van attributen en constanten. Zo kan bijvoorbeeld ook hier een variabele met domein ALFANUMERIEK niet in een rekenkundige expressie gebruikt worden.
- variabelen bij opvragingen omtrent elk willekeurig samengesteld type gebruikt kunnen worden. Dit geldt natuurlijk ook voor constanten, maar zeker niet voor attributen.

Hieronder nog een tweetal voorbeelden:

```
value maxprijs = max artikel its prijs.
value maxprijsmin175ob = maxprijs / 1.175.
```

Bewaar de prijs van het duurste artikel exclusief 17,5% omzetbelasting in variabele ’maxprijsmin175ob’.

```
value totomzet =
        total verkoop its bedrag
        where datum >= datef(19920301).
```

```
value totverkoop =
        total verkoop its hoeveelheid
        where datum >= datef(19920301).
value gemprijs = totomzet / totverkoop.
```

Bepaal de gemiddelde prijs waarvoor produkten vanaf 1 maart 1992 zijn verkocht en bewaar dit in 'gemprijs'.

In de eerste value-opdracht wordt de omzet vanaf 1 maart 1992 bepaald, deze waarde wordt in 'totomzet' bewaard. In de tweede value-opdracht wordt vervolgens het totaal aantal produkten dat vanaf 1 maart 1992 verkocht is bepaald, dit aantal wordt in 'totverkoop' geplaatst. Deling van totomzet door totverkoop levert tenslotte de gemiddelde prijs waarvoor artikelen vanaf 1 maart 1992 zijn verkocht.

III.5. EXTEND-OPDRACHT

III.5.1. Inleiding

De extend-opdracht wordt gebruikt om een bepaald samengesteld type **tijdelijk** met een attribuut uit te breiden. Het attribuut waarmee een samengesteld type tijdelijk uitgebreid wordt heet de **extensie**. Een extensie is een tijdelijk attribuut omdat het niet permanent in het conceptuele model wordt opgenomen. Een extensie wordt verwijderd indien:

- query verlaten wordt. In dat geval worden altijd alle extensies verwijderd.
- men hiertoe opdracht geeft. Zie hiervoor paragraaf VI.2.11. DELEXT.

De waarde van een extensie wordt afgeleid uit de opgeslagen gegevens. Een extensie bevat dus redundante data. Zo heeft het geen zin bij afdeling een attribuut aantal werknemers op te nemen omdat dit afleidbaar is door telling.

Aan extensies worden drie eisen gesteld, te weten:

- Een extensie moet altijd een (eventueel nieuw) basistype zijn, dus nooit samengesteld. Dit is zo gekozen omdat we niet willen dat opdrachten het conceptuele model met nieuwe samengestelde typen kunnen uitbreiden.

- Een extensie mag **geen** prefix bevatten. Met andere woorden: een extensie kan nooit een rol-attribuut zijn. Dat is zo gekozen omdat een extensie gezien moet worden als een klad-attribuut dat louter tussenresultaten bevat die een rol spelen bij een bepaalde opvraging. Na beëindiging van de opvraging kan de extensie in principe weer worden verwijderd. Het zou overdreven zijn om een extensie, die geen wezenlijke rol in het conceptuele model speelt, als rol-attribuut op te nemen.

- De naam van de extensie is beperkt. De beperkingen zijn:
  - het moet beginnen met een letter, gevolgd door een aantal letters en/of cijfers.
  - de naam van de extensie moet uniek zijn binnen de attributen van het type dat met deze extensie wordt uitgebreid. Dezelfde naam mag wel elders in het conceptuele model gebruikt worden.
  - de naam van de extensie niet gelijk mag zijn aan standaardwoorden zoals **'get', 'its'** en dergelijke.

Een extensie is dus een tijdelijk basistype zonder prefix, dat een tijdelijk attribuut wordt van een bepaald samengesteld type. Een extensie wordt gedefinieerd met behulp van de extend-opdracht. Bij het definitie van de extensie wordt direct een waarde gegeven aan het tijdelijke attribuut. Net als met variabelen en normale basistypen geldt dat een extensie maar **één** waarde bevat. Analoog aan de value-opdracht wordt daarom ook hier onderscheid gemaakt tussen de extend-opdracht met en zonder set-functie. De variant met set-functie wordt gebruikt om daar waar nodig af te dwingen dat een bepaald resultaat maar **één** waarde heeft.

III.5.2. Extensie zonder set-functie

In dit geval heeft men dus te maken met een extend-opdracht waarbij het resultaat automatisch altijd uit **één** waarde bestaat. Bijvoorbeeld:

> **extend** verkoop **with** bedragex175ob = bedrag / 1.175.
>
> Voeg aan verkoop de extensie bedragex175ob toe. "Bedragex175ob" is de aanduiding van de verkoop exclusief 17,5% omzetbelasting.

Het samengesteld type dat direct op '**extend**' volgt, is het type dat wordt uitgebreid. De naam die op '**with**' volgt, is de naam van de nieuw toe te voegen extensie. Het gedeelte dat op '=' volgt, specificeert de waarde die de nieuwe extensie moet krijgen.

Het gedeelte dat specificeert welke waarde de nieuwe extensie moet krijgen werkt in het geval van een extend-opdracht zonder set-functie analoog aan de attribuut selectie die bij de get-opdracht gebruikt wordt. Men kan hier echter maar **één** attribuutpad specificeren. Hierbij is het type volgens welke het conceptuele model wordt benaderd, in dit voorbeeld verkoop, weer het access-type (AT).

Met behulp van de eerder gegeven extend-opdracht is verkoop dus uitgebreid met extensie bedragex175ob. Deze extend-opdracht geeft echter nog niet het resultaat van deze extensie. Dit resultaat wordt verkregen door een get-opdracht op het type verkoop uit te voeren, bijvoorbeeld:

> **get** verkoop.
>
> Haal alle gegevens van alle verkopen.

Het resultaat van deze get-opdracht bevat naast de identificatie, alle attributen van het type verkoop. Eén van deze attributen is de extensie bedragex175ob.

De extend-opdracht kan dus tijdelijke attributen aan samengestelde typen toevoegen, maar het resultaat van een extend-opdracht moet met de get-opdracht worden opgevraagd.
Natuurlijk is het resultaat van de extend-opdracht uit het voorbeeld niet echt indrukwekkend, dit is immers ook direct op te vragen:

get verkoop **its** bedrag / 1.175.

Haal van iedere verkoop het verkoopbedrag exclusief 17,5% omzetbelasting.

Ieder ander gewenst attribuut van verkoop moet nu echter wel specifiek genoemd worden, maar dat is natuurlijk geen reden om in dit geval extensies te gebruiken. Kort gezegd komt het erop neer dat het resultaat van iedere extend-opdracht zonder set-functie ook direct met behulp van een get-opdracht is op te vragen. Dit soort extend-opdrachten heeft dan ook alleen zin indien het resultaat ervan, de extensie dus, verschillende malen nodig is in de opvragingen. Neem het voorbeeld dat de artikelen die met meer dan fl. 100,- korting verkocht zijn, tezamen met de korting die per artikel gegeven is, gevraagd worden. Om aan deze opvraging te voldoen moet bij gebruik van een get-opdracht tweemaal dezelfde verleende korting op de verkochte artikelen berekend worden, namelijk eenmaal bij attribuut en eenmaal bij instance selectie. Indien gebruik wordt gemaakt van een extend-opdracht is dit tot éénmaal te reduceren, namelijk:

extend verkoop **with** korting =
    (hoeveelheid * verkart **its** artikel **its** prijs - bedrag) / hoeveelheid.
get verkoop **its** korting, verkart **its** artikel
    where korting > 100.

Het blijkt aldus dat de extensie direct na specificatie te gebruiken is in verdere opdrachten. Het gebruik van een extensie is analoog aan het gebruik van normale attributen van een bepaald samengesteld type. Daar zijn al voorbeelden van gegeven bij de get-opdracht.

Kenmerkend voor dit type extend-opdracht is dat gegevens, benodigd voor de extensie, steeds van een niveau worden gehaald dat niet hoger is dan het access-type in het conceptuele model. Dit is tevens de reden waarom het hier zonder set-functie mogelijk is. De gegevens uit de ontledingsboom van het access-type zijn immers enkelvoudig, d.w.z. iedere instance van het access-type heeft **precies één** waarde voor de attributen waaruit het is samengesteld. **Precies één** vanwege de relateerbaarheid.

Beschouw verder de volgende extend-opdracht:

extend verkoop **with** prijs = verkart **its** artikel **its** prijs
    where verkart **its** artikel **its** prijs > 1000.

Deze extend-opdracht is **niet correct** vanwege de toegepaste instance selectie. Hierdoor zouden namelijk alleen de extensies van instances die geselecteerd worden een waarde krijgen. Van instances die niet geselecteerd worden blijft de waarde van de extensie ongedefinieerd, hetgeen moet worden voorkomen. Een where-clausule in een extend zonder set-functie is dus niet toegestaan.

III.5.3. Extensie met set-functie

De werking van deze opdracht is in grote mate overeenkomstig de get/value-opdracht met set-functie. Het verschil is echter dat nu voor iedere instance van het te beschouwen samengesteld type een waarde voor de gespecificeerde set-functie wordt geëvalueerd. Eerst een voorbeeld:

extend afdeling **with** aantwerkn =
count werknemer
per afdeling.

Breid afdeling uit met extensie ’aantwerkn’, het aantal werknemers dat op de betreffende afdeling werkt.

In tegenstelling tot de extend-opdracht zonder set-functie worden hier gegevens vanaf een **hogere** plaats in het conceptuele model dan het access-type naar beneden te gehaald. Bij de extend-opdracht zonder set-functie heeft men altijd te maken met de ontledingsboom van het access-type. Hier echter speelt de ontledingsboom van een samengesteld type dat hoger dan het access-type in het conceptuele model ligt een rol. Dit tweede type noemen we verder het **extensie-type** (ET). Dit ET ligt in de nesting van het AT. In dit voorbeeld is ’afdeling’ het AT en ’werknemer’ het ET.

Het feit dat het ET in de nesting van het AT ligt geeft aan dat:
- het AT in de ontledingsboom van het ET ligt. Het attribuutpad van ET naar AT moet m.b.v. het ’**per**’-gedeelte worden gespecificeerd. In dit voorbeeld dus: ’**per** afdeling.’. Dit moet gedaan worden omdat er in het conceptuele model meer dan één pad van ET naar AT kan lopen. Dit zal met name het geval zijn indien er sprake is van rol-attributen.

- er bij **één** instance van het AT een willekeurig aantal instances van het ET kan voorkomen. Op een bepaalde afdeling kan immers een willekeurig aantal werknemers werken. Juist omdat in de extensie op een bepaalde afdelings-instance maar **één** waarde wordt opgeborgen moet in dit geval met set-functies worden gewerkt.

De werking van de gegeven extend-opdracht is als volgt:

1) Allereerst wordt de extensie ’aantwerkn’ op het type afdeling gemaakt. Deze extensie krijgt voor iedere instance initieel de waarde 0.

2) Vervolgens wordt het attribuut afdeling van alle werknemer-instances in ogenschouw genomen, en wordt bij de betreffende instance van afdeling de waarde van attribuut 'aantwerkn' met 1 opgehoogd.

Met behulp van de extend-opdracht met set-functie is men in staat een groot aantal nieuwe soorten van opvragingen uit te voeren. De opvraging van de afdeling waar de meeste werknemers werken verloopt nu bijvoorbeeld als volgt:

```
extend afdeling with aantwerkn =
        count werknemer
        per afdeling.
value maxwerkn = max afdeling its aantwerkn.
get afdeling where aantwerkn = maxwerkn.
```

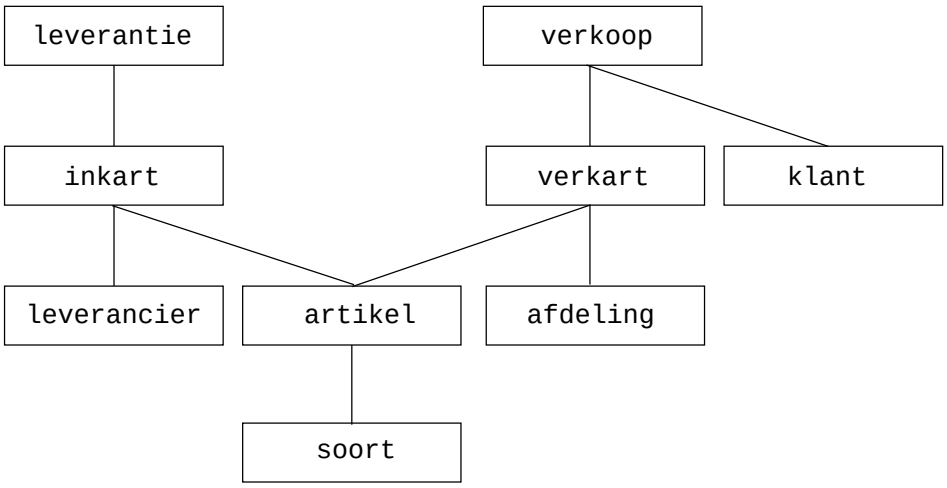
In de extend-opdracht zonder set-functie kon bij attribuut en instance-selectie gebruik gemaakt worden van alle attributen in de ontledingsboom van het AT. Hier echter kan er gebruik gemaakt worden van alle attributen in de ontledingsboom van het ET. Dat het domein van toegestane attributen aldus toeneemt blijkt uit het feit dat de ontledingsboom van het ET de ontledingsboom van het AT omvat. Het AT zit immers in de ontledingsboom van het ET. Bijvoorbeeld:

```
extend klant with speelgoedaankopen =
        count verkoop
        where verkart its artikel its soort = "speelgoed"
        per klant.
```

Breid klant uit met extensie 'speelgoedaankopen', het aantal aankopen van artikelen van de soort speelgoed per klant.

In dit geval is het AT klant en het ET verkoop. Duidelijk is dat we in dit geval voor instance selectie gebruik maken van attributen in de ontledingsboom van het ET. Per klant worden in deze opvraging eerst de instances van verkoop geteld waarvoor het verkochte artikel van de soort speelgoed is. Vervolgens wordt dit aantal in extensie speelgoedaankopen van de betreffende klant instance opgeslagen.

Tot dusver zijn alleen voorbeelden gegeven van opvragingen waarbij de gewenste informatie kon worden afgeleid m.b.v. **één** extend-opdracht waarbij het AT een direct attribuut van het ET was. Er zijn echter ook opvragingen waarbij het attribuutpad van ET naar AT langs andere samengestelde typen loopt.



Een deel van de abstractiehiërarchie van de warenhuis-database.

Stel bijvoorbeeld dat men geïnteresseerd is in de omzet van iedere afdeling. Uit het conceptuele model blijkt dat deze opvraging betrekking heeft op de samengestelde typen verkoop en afdeling. In verkoop staan immers de bedragen waarvoor artikelen zijn verkocht, en afdeling is het type waarvoor de omzet gewenst wordt. Om deze gewenste omzet per afdeling te kunnen afleiden is dus kennelijk een extend-opdracht nodig met AT afdeling en ET verkoop. Nu blijkt echter dat het type verkart op het attribuutpad van verkoop naar afdeling ligt, zodat de tot dusver gebruikte extend-opdracht met het AT een direct attribuut van het ET hier geen uitkomst biedt. Het hier gestelde probleem kan op twee manieren opgelost worden.

Oplossing 1:

Het probleem omzeilen. Dit is mogelijk door de gewenste extensie op afdeling in twee stappen uit te voeren. Ten eerste een extensie op verkart en vervolgens een extensie op afdeling. Men heeft dan twee extend-opdrachten die er qua structuur hetzelfde uitzien als de extend-opdrachten in voorgaande voorbeelden. De extensie op verkart ziet er als volgt uit:

extend verkart **with** omzet =
    total verkoop **its** bedrag
    per verkart.

Breid verkart uit met extensie omzet, de omzet van de betreffende instance van verkart.

Extensie omzet bevat dus per instance van verkart de omzet van het betreffende artikel op de betreffende afdeling. Vervolgens is de omzet per afdeling te bepalen door de omzet van artikelen per afdeling te sommeren:

extend afdeling **with** omzet =
total verkart **its** omzet
per afdeling.

Breid afdeling uit met extensie omzet, de totale omzet van de betreffende instance van afdeling.

Oplossing 2:

De extend-opdracht aanpassen. De aanpassing die aan de extend-opdracht gemaakt is om aan dit probleem het hoofd te bieden is in feite vrij triviaal. Men neemt gewoon net als bij voorgaande voorbeelden van de extend-opdracht in het per-gedeelte het attribuutpad van ET naar AT op. Als enige verschil loopt het attribuutpad in dit geval eerst nog via andere samengestelde typen. De oplossing van het probleem is dus ook als volgt mogelijk:

extend afdeling **with** omzet1 =
total verkoop **its** bedrag
per verkart **its** afdeling.

Dat het resultaat van beide oplossingen gelijk is kunnen we controleren m.b.v.:

get afdeling **where** omzet <> omzet1.

De lengte van dit attribuutpad van ET naar AT is natuurlijk afhankelijk van enerzijds het conceptuele model en anderzijds hetgeen men wil opvragen. Hieronder een voorbeeld voor het gegeven conceptuele model waarbij de lengte van dit attribuutpad maximaal is:

extend soort **with** verkocht =
any verkoop
per verkart **its** artikel **its** soort.

Breid soort uit met extensie verkocht. Deze extensie krijgt de logische waarde TRUE indien er verkopen zijn geweest van artikelen van de betreffende soort.

In deze extend-opdracht wordt per soort artikel gekeken of er instances van het type verkoop zijn waarbij het verkochte artikel van deze soort is.
Deze laatste extend-opdracht is als volgt op te splitsen:

extend verkart **with** verkocht =
any verkoop **per** verkart.
extend artikel **with** verkocht =
any verkart
where verkocht
per artikel.

```
extend soort with verkocht =
        any artikel
        where verkocht
        per soort.
```

III.5.4. Representatie van een extensie

Het domein en de representatie wordt door Xplain afgeleid uit de expressie-expressie. Het type van een extensie levert hierbij nooit een probleem voor de gebruiker op. Daarom hoeft de gebruiker normaliter ook geen domein te specificeren. In enkele gevallen kan de afgeleide representatie echter tot overflow leiden. In dat geval is het mogelijk de representatie die door Xplain bepaald wordt te overtreffen. De volgende opvraging is hiervan een voorbeeld:

```
extend soort with omzet (r9,2) =
        total verkoop its bedrag
        per verkart its artikel its soort.
get soort its beschrijving, omzet.
```

III.6. VOORBEELDEN

Hier worden een aantal opvragingen beschreven waarvoor een combinatie van get-, value- en extend-opdrachten nodig is om het gewenste resultaat te bereiken.

Voorbeeld 1: Welke artikelen worden door alle leveranciers geleverd?

Voor de artikelen die hieraan voldoen geldt dat de verzameling van leveranciers die het artikel leveren gelijk is aan de verzameling van leveranciers. Het aantal leveranciers dat het artikel levert moet dus gelijk zijn aan het aantal leveranciers. Het aantal leveranciers wordt bepaald d.m.v.:

```
        value aantlev = count leverancier.
```

Het aantal leveranciers dat een artikel levert is via inkart te bepalen. Dit wordt gerealiseerd door per artikel het aantal instances van inkart te tellen. Immers, wegens omkeerbaarheid kan er hoogstens één instance van inkart voorkomen met een bepaalde waarde voor artikel en leverancier. Dus door het aantal instances van inkart met gelijke waarde van artikel te tellen krijgt men het aantal verschillende leveranciers die het betreffende artikel leveren. Men krijgt aldus:

```
extend artikel with aantal =
        count inkart
        per artikel.
```

Merk op dat deze formulering onjuist is als inkart naast artikel en leverancier nog meer attributen zou hebben.

De artikelen die door alle leveranciers geleverd worden zijn nu als volgt op te vragen:

get artikel **where** aantal = aantlev.

Voorbeeld 2: Welke leveranciers leveren tenminste dezelfde artikelen als leverancier ’LO3’?

Voor de leveranciers die hieraan voldoen geldt dat het aantal artikelen dat door de leverancier ’LO3’ **en** betreffende leverancier geleverd wordt gelijk moet zijn aan het aantal artikelen dat leverancier ’LO3’ levert. De artikelen die leverancier ’LO3’ levert zijn als volgt in de database aan te duiden:

extend artikel **with** LO3leverbaar =
any inkart
where leverancier = "LO3"
per artikel.

De instances van artikel waarbij het artikel door leverancier ’LO3’ geleverd wordt hebben nu dus de logische waarde TRUE voor extensie LO3 leverbaar. Van elk artikel is nu bekend of het wel of niet van leverancier ’LO3’ afkomstig is.

Vervolgens is per leverancier het aantal artikelen te bepalen dat ook door leverancier ’LO3’ geleverd wordt:

extend leverancier **with** aantLO3art =
count inkart
where artikel **its** LO3leverbaar
per leverancier.

In aantLO3art staat nu dus per leverancier het aantal artikelen dat zowel door de leverancier als door leverancier ’LO3’ verkocht wordt.

Het aantal artikelen dat leverancier ’LO3’ verkoopt wordt bepaald door:

value aantalLO3 = **count** artikel **where** LO3leverbaar.

Of door:

value aantalLO3 = **count** inkart **where** leverancier = "LO3".

Tenslotte worden nu de leveranciers opgevraagd die **minstens** dezelfde artikelen als leverancier ’LO3’ leveren via:

get leverancier **where** aantLO3art = aantalLO3.

Voorbeeld 3: Welke leveranciers leveren geen van de artikelen die leverancier 'LO3' levert?

Deze leveranciers zijn op te vragen door na de twee extend-opdrachten uit voorbeeld 2 direct de volgende get-opdracht uit te voeren:

get leverancier **where** aantLO3art = 0.

Voorbeeld 4: Welke leveranciers leveren **precies** dezelfde artikelen als leverancier 'LO3'?

Dit resultaat wordt verkregen door bij de opdrachten uit voorbeeld 2 toe te voegen:

extend leverancier **with** aantart =
count inkart
per leverancier.

Extensie aantart bevat dus het aantal artikelen dat betreffende leverancier levert. Het gewenste resultaat wordt verkregen door de get-opdracht uit voorbeeld 2 te veranderen in:

get leverancier
where aantLO3art = aantalLO3 **and** aantart = aantLO3art.

III.7. INSERT-OPDRACHT

De insert-opdracht wordt gebruikt om instances van een bepaald samengesteld type in de database in te voeren. Hierbij moet met het volgende rekening gehouden worden:

- ieder attribuut van de nieuw toe te voegen instance moet via de insert-opdracht een waarde krijgen. Dit geldt zowel voor de identificatie als voor de attributen. Een attribuut waar echter een **init**-restrictie voor gedefinieerd is, wijkt hiervan af omdat de waarde door het DBMS bepaald wordt volgens de init-expressie. Een attribuut met **init**-restrictie mag dus **niet** voorzien worden van een meegegeven waarde. Een attribuut met **init default**-restrictie mag daarentegen **wel** voorzien worden van een meegegeven waarde. De mee-gegeven waarde zal dan gebruikt worden. Als er geen waarde meegegeven wordt dan zal het DBMS de default waarde gebruiken zoals gespecificeerd in de init-expressie.

- de identificatie van de nieuw toe te voegen instance moet uniek zijn binnen de collectie identificaties van de instances van het betreffende samengestelde type.

- voor attributen die een samengesteld type zijn moet de identificatie van het betreffende samengestelde type ingevoerd worden. Dit mogen alleen identificaties zijn van instances

die reeds in de database zijn opgenomen (relateerbaarheid).

- voor attributen die tot een basistype behoren moeten waarden ingevoerd worden die voldoen aan het voor de betreffende attributen vereiste formaat en eventuele waardebeperking.

Enkele voorbeelden:

insert soort "computer" **its** beschrijving = "soft- / hardware".

Voeg een instance van samengesteld type soort toe met identificatie ’computer’ en beschrijving ’soft- / hardware’.

value soortnaam = **input** (a8) "Geef de soortnaam : ".
value soortbeschrijving = **input** (a16) "Geef de soortbeschrijving: ".
insert soort soortnaam **its** beschrijving = soortbeschrijving.

Deze opdrachten maken het mogelijk om gegevens in te voeren volgens eigen dialoog.

insert artikel "1234567" **its**
beschrijving = "handleiding",
kleur = "paars/groen",
voorraad = 100,
prijs = 87.50,
soort = "computer".

Voeg een instance van samengesteld type artikel toe met identificatie ’1234567’.

Eenvoudig valt te controleren dat deze voorbeelden voldoen aan de eerder opgesomde eisen t.a.v. het toevoegen van instances. Men kan overigens de volgorde van deze twee insert-opdrachten niet veranderen omdat op het moment dat artikel ’1234567’ toegevoegd wordt, soort ’computer’ reeds moet bestaan. Dit betreft natuurlijk de relateerbaarheid.

Het is daarnaast mogelijk om de nieuwe unieke instance-identificatie door Xplain DBMS zelf te laten bepalen door ’*****’ te gebruiken. Xplain DBMS bepaalt in dit geval zelf een nieuwe unieke identificatiewaarde. Een voorbeeld is de volgende toevoeging van een klant.

insert klant *** its**
naam = "Jansen",
voorl = "J",
adres = "Brederodelaan 24",
postc = "7312NJ",
woonplaats = "Apeldoorn",
schuld = 0.

III.8. DELETE-OPDRACHT

De delete-opdracht wordt gebruikt om instances uit de database te verwijderen. Het selecteren van te verwijderen instances gaat analoog aan het selecteren van instances bij de get-opdracht. Enkele voorbeelden:

delete artikel "1234567".

Verwijder het artikel met instance identificatie ’1234567’.

delete artikel **where** soort = "computer".

Verwijder alle artikelen van de soort "computer".

value soortnaam = **input** (a8) "Geef de soortnaam: ".
delete soort soortnaam.

Verwijder soort met de ingegeven soortnaam.

Bij het verwijderen van instances moet er rekening mee gehouden worden dat dit **alleen** mogelijk is indien betreffende instance niet elders als attribuut in de database voorkomt. Het is bijvoorbeeld niet mogelijk om de instance ’computer’ van samengesteld type ’soort’ uit de database te verwijderen indien er nog artikelen van de soort "computer" opgeslagen zijn. Anders zou het principe van relateerbaarheid aangetast worden.

III.9. UPDATE-OPDRACHT

De update-opdracht wordt gebruikt om attribuutwaarden van opgeslagen instances in de database te veranderen. Selectie van instances waarvan de attributen moeten worden aangepast gaat op dezelfde wijze als de instance selectie bij de get-opdracht. Enkele voorbeelden:

update artikel "1234567" **its** prijs = 97.50.

Verander de prijs van artikel ’1234567’ in fl. 97.50.

update artikel **its** prijs = 0.90 * prijs
where soort = "computer".

Verlaag de prijs van alle artikelen van de soort computer met 10%.

Met behulp van de update-opdracht zijn alleen de directe attributen van het access-type te veranderen. De volgende update opdracht is dus **niet correct**:

update verkart **its** artikel **its** prijs = 10 * artikel **its** prijs.

Wel is in de update-opdracht een willekeurige instance selectie te gebruiken. Ook is meer dan één attribuut van het access-type tegelijk te veranderen, bijvoorbeeld:

update verkoop **its**
bedrag = .95 * bedrag,
hoeveelheid = 1.10 * hoeveelheid,
aanbet = aanbet + 3000
where klant **its** naam = "Pieterse"
and verkart **its** artikel **its** soort = "computer".

Verander een aantal gegevens van instances van verkoop waarbij de klant "Pieterse" is en de verkochte artikelen van de soort computer zijn.

In deze voorbeelden worden alleen attributen veranderd die een basistype zijn. Het is echter ook mogelijk om attributen die een samengesteld type zijn te veranderen. Een werknemer kan bijvoorbeeld naar een andere afdeling worden overgeplaatst. Men moet er dan echter wel op letten dat deze nieuwe attribuutwaarde reeds in de database is opgenomen als de instance identificatie van een instance van het betreffende type. Met andere woorden, de afdeling waar de werknemer naar wordt overgeplaatst moet al in de database zijn opgenomen. Bijvoorbeeld:

update werknemer "9" **its** afdeling = "20".

Plaats werknemer 9 over naar afdeling 20.

Verder kent de update-opdracht nog twee beperkingen:

1 de identificatie van een instance mag niet veranderd worden. Dit is zo gekozen omdat de betreffende instance als attribuut kan voorkomen van instances van typen die hoger in het model liggen. Een verandering van de identificatie zou dus een verbreking van de relateerbaarheid tot gevolg kunnen hebben. Tevens geldt dat een identificatie geen beschrijvend attribuut maar een onafhankelijk uniek identificerend kenmerk dat in principe niet zou moeten veranderen gedurende de levensduur van de instance.

2 een update-opdracht mag nooit het domein van een type veranderen. Men wil immers niet dat het gebruik resulteert in veranderingen van het conceptuele model van de database.

III.10. CASCADE-OPDRACHT

De cascade opdracht speelt een cruciale rol in de formulering van **recursieve problemen**. Hiermee zijn tal van complexe vraagstukken te specificeren met behulp van de querytaal.

Voorbeelden van recursie vinden we in ondermeer bij:
- kritieke pad analyse (om bijvoorbeeld de minimale doorlooptijd te bepalen van een project bestaande uit diverse activiteiten).
- stambomen (om bijvoorbeeld de voorouders of de nakomelingen van een persoon te bepalen).
- stuklijsten (om bijvoorbeeld de samenstelling van producten en benodigde bestellingen in zo'n situatie te bepalen).

Een eenvoudige toepassing van de cascade-opdracht treffen we in het volgende voorbeeld. Veronderstel een eenvoudig semantisch model met de volgende typen:

```
type knoop = naam.
type verbinding = van_knoop, naar_knoop.
```

Hierbij kan de zogenaamde topologische ordening van knopen als volgt bepalen:

```
extend knoop with niveau = 0.
cascade knoop its niveau =
        max verbinding its van_knoop its niveau + 1
        per naar_knoop.
get knoop its naam, niveau.
```

Een uitvoerige verhandeling van de cascade-opdracht kan gevonden worden in recente publicaties over semantische databases.

III.11. SPECIALISATIE

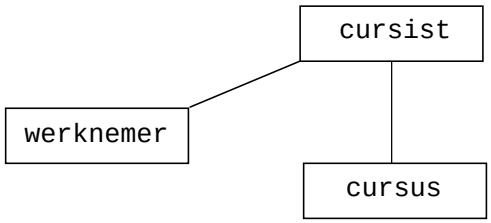
Tot dusver is er nog geen aandacht besteed aan specialisatie. Omdat dit wel degelijk een belangrijk concept is wordt er het volgende voorbeeld aan gewijd. In dit voorbeeld wordt een uitbreiding aangebracht op de warenhuis-database.

Het geval wil namelijk dat in het kader van een algemeen opleidingsbeleid, de werknemers deel kunnen nemen aan maximaal **één** van de door het bedrijf verzorgde cursussen. Maximaal één vanwege de beperkte capaciteit. De cursussen worden gedurende een bepaalde periode in de avonduren gegeven. Iedere cursus wordt steeds op dezelfde weekdag in steeds dezelfde zaal gegeven.

Het conceptuele model voor opslag van deze gegevens ziet er als volgt uit:

```
type cursus     = begin_datum, eind_datum, dagnaam, zaal.
type cursist    = [werknemer], cursus.
```

Dit geeft het volgende plaatje:



Hierbij is cursist een specialisatie van werknemer. Iedere werknemer kan immers deelnemen aan ten hoogste één cursus. Het feit dat cursus een attribuut van cursist is weerspiegelt dat iedere cursist deelneemt aan een cursus, en dat er meerdere cursisten aan dezelfde cursus kunnen deelnemen. De naam van de cursus wordt direct in de identificatie van cursus geplaatst. Hier is elke cursusnaam uniek.

Nu de specialisatie is geïntroduceerd wordt begonnen met de behandeling van een aantal relevante opvragingen. Ten eerste wordt nader bekeken wat het resultaat is van een get-opdracht op het type cursist:

get cursist.

Het resultaat van deze get-opdracht is een tabel waarin de gegevens van alle instances van het type cursist voorkomen. Deze tabel bestaat per instance uit respectievelijk de identificatie van de cursist, de identificatie van de corresponderende werknemer en de waarde van het attribuut cursus (= cursus-identificatie).

Slechts de hier genoemde generalisatie behoeft nader te worden toegelicht. Welnu, deze generalisatie bestaat louter uit de identificatie van de betreffende werknemer. Bij ieder instance van cursist moet immers een instance van werknemer zijn opgenomen (relateerbaarheid).

Het opvragen van de attributen van de specialisatie gaat precies hetzelfde als bij de reeds behandelde voorbeelden met aggregatie. Bijvoorbeeld:

get cursist **its** werknemer **its** naam.

Hier worden de namen opgevraagd van alle werknemers die cursist zijn.

Men kan hier opmerken dat in dit model cursist een synoniem voor werknemer is en dat van ’cursist **its** naam’ gesproken kan worden. Om reden van consistentie wordt werknemer toch opgevat als attribuut van cursist.

Hiermee is het geval behandeld waarin vanuit de specialisatie naar de generalisatie wordt gerefereerd. Dit gaat in feite precies hetzelfde als bij aggregatie. Alleen moeten we nu met **its** gaan werken indien men de attributen van het gegeneraliseerde type wil adresseren. Men

maakt hierbij steeds gebruik van de relateerbaarheid tussen de specialisatie en de generalisatie. Bij iedere opgeslagen instance van het type cursist moet immers een instance van het type werknemer aanwezig zijn.

Indien men echter vanuit de generalisatie de bijbehorende specialisatie wil adresseren moet men anders te werk gaan. Stel bijvoorbeeld dat men wil weten aan welke cursus de werknemers deelnemen. Dit is als volgt te bereiken:

```
extend werknemer with cursus =
        some cursist its cursus
        per werknemer.
get werknemer.
```

Eerst wordt het type werknemer uitgebreid met de extensie cursus. Deze extensie krijgt de naam van de cursus die de betreffende werknemer volgt. Indien een werknemer geen cursus volgt dan krijgt extensie cursus een zogenaamde verstekwaarde (defaultwaarde). Deze defaultwaarde moet niet opgevat worden als een waarde waarop geselecteerd kan worden. Het enige dat deze defaultwaarde aangeeft is dat de gewenste waarde, vanwege het niet voorkomen van de specialisatie, voor deze instance van de generalisatie niet gedefinieerd is.
De volgende opvraging is dus **niet correct**:

```
get werknemer where cursus = "undefined".
```

Hierbij moet ’undefined’ opgevat worden als de speciale waarde met bovengenoemde betekenis.
Wat men met een opvraging als hierboven eigenlijk wil is een overzicht met daarin de instances van het gegeneraliseerde type die niet in de specialisatie voorkomen. Dit wordt als volgt opgevraagd:

```
extend werknemer with nocursist =
        nil cursist
        per werknemer.
get werknemer where nocursist.
```

De bovenstaande voorbeelden zijn typerend voor het opvragen van gegevens betreffende een specialisatie vanuit het gegeneraliseerde type. Men gebruikt hierbij steeds eerst een extend-opdracht met set-functie. Het is echter niet altijd noodzakelijk een extensie te maken. Beschouw het volgende voorbeeld. Welke werknemers hebben op donderdag cursus?
Oplossing 1:

```
extend werknemer with donderdagcursist =
        any cursist where cursus its dag = "do"
        per werknemer.
get werknemer where donderdagcursist.
```

Oplossing 2:

```
get cursist its werknemer
    where cursus its dag = "do".
```

Het enige verschil met de voorgaande oplossing is dat men nu ieder gewenst attribuut van werknemer afzonderlijk moet specificeren, terwijl bij de eerste oplossing direct alle attributen gegeven worden. Verder wordt bij de eerste oplossing een extensie op werknemer gemaakt, dit hoeft echter niet altijd noodzakelijk te zijn.

In dit voorbeeld komt in feite de afweging omtrent het al dan niet maken van extensies zoals eerder gegeven bij de behandeling van extensies terug. Hierbij valt op te merken dat alleen indien het model wordt benaderd vanuit de generalisatie een extensie nodig is. Men gaat dan immers omhoog in de abstractiehiërarchie.

Uit het voorgaande moet overigens niet afgeleid worden dat de keuze tussen het al dan niet maken van een extensie volstrekt willekeurig is. Met name het geval waarin het resultaat zoals dit m.b.v. een get-opdracht kan worden opgevraagd als tussenresultaat opgevat moet worden, is hiervan een voorbeeld. Beschouw het volgende voorbeeld.

Werknemers die deelnemen aan de cursus brandbeveiliging gaan ter introductie een dag bij de plaatselijke brandweer kijken. Opdat de afdelingshoofden van de afdelingen waarop de betreffende werknemers werken niet voor verrassingen komen te staan, wordt er prijs op gesteld dat deze een berichtje krijgen over het aantal werknemers dat zij die dag moeten missen.

```
extend werknemer with brandbev =
    any cursist
    where cursus = "brandbeveiliging"
    per werknemer.
extend afdeling with afwezig =
    count werknemer
    where brandbev
    per afdeling.
get afdeling.
```

Hoewel het resultaat van de eerste extend-opdracht ook met een get-opdracht kan worden opgevraagd wordt hiervoor toch een extensie gemaakt, omdat het betreffende resultaat nodig is als tussenresultaat in de opvraging.

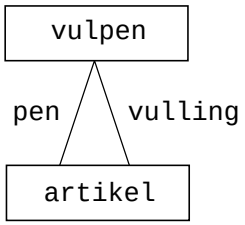
III.12. ROL-ATTRIBUTEN

In deze paragraaf wordt een uitbreiding aangebracht op het conceptuele model van de

warenhuis-database, ten einde een voorbeeld te presenteren waarin het gebruik van rol-attributen met **prefixen** aan de orde gesteld wordt.

Het betreft hier een probleem waarmee de afdeling kantoorbenodigdheden stoeit. Als gevolg van een uitbreiding van het assortiment vulpennen en vullingen dreigt het personeel het overzicht hierover te verliezen. Gevraagd wordt een hulpmiddel om te kunnen bepalen welke vullingen geschikt zijn voor welke vulpennen en andersom.

Ten einde genoemde afdeling een handje te helpen zou de volgende uitbreiding op het conceptuele model kunnen worden aangebracht:



Rolattributen

type vulpen = pen_artikel, vulling_artikel.

In dit type wordt iedere mogelijke combinatie van penhouder en vulling vastgelegd. Zowel pen_artikel als vulling_artikel bevatten de identificatie van een reeds in de database opgenomen instance van het type artikel. De prefixen **pen** en **vulling** geven aan welke rol de beide artikelen in het type vulpen hebben.

In de tekening worden de verschillende rollen die het type artikel in het type vulpen speelt gerepresenteerd door voor iedere rol een afzonderlijke verbinding tussen de betreffende typen aan te brengen.

Als eerste relevante opvraging: alle vullingen die in het assortiment zijn opgenomen voor de pen met identificatie ’1234567’:

get vulpen **its** vulling_artikel **where** pen_artikel = 1234567.

Het resultaat hiervan is een tabel met alle identificaties van het type artikel die aan het gestelde voldoen.

Verder is het zinvol om vullingen die slechts voor **één** type pen geschikt zijn altijd bij dit type pen te leveren. De vullingen die hier aan voldoen zijn als volgt op te sporen:

extend artikel **with** npen =
count vulpen **per** vulling_artikel.

Het resultaat van deze opdracht kan het beste toegelicht worden m.b.v. een beschrijving van de verwerkingen die deze tot gevolg heeft. Welnu, allereerst maakt deze opdracht de extensie npen op artikel en initialiseert deze extensie voor alle instances van artikel met de integerwaarde 0. Vervolgens worden alle instances van vulpen bezien. Elke keer dat vanuit vulpen via het attribuut vulling_artikel een artikel wordt aangewezen, wordt bij dat artikel de waarde van npen met 1 opgehoogd. Duidelijk is dat de artikelen die geen vulling_artikel zijn de waarde 0 voor npen behouden. Deze artikelen zijn immers nooit als vulling_artikel in een instance van vulpen opgenomen.

Het voorbeeld wordt afgerond met de volgende opdracht:

```
get artikel where npen = 1.
```

De instances van artikel die slechts éénmaal als vulling_artikel in een instance van vulpen voorkomen en dus slechts voor **één** type pen geschikt zijn, worden uit de database gelicht.

Als laatste voorbeeld het volgende: welke penhouders zijn er in het assortiment zonder bijpassende vulling?

De oplossing hiervan is als volgt:

```
extend artikel with geenvulling =
        nil vulpen per pen_artikel.
get artikel
        where geenvulling and beschrijving = "penhouder".
```

Toelichting:

"**get** artikel **where** geenvulling" bevat een onvoldoend aantal voorwaarden. Hiermee worden ook artikelen verkregen die geen penhouder zijn, bijvoorbeeld een mes of vork.

III.13. PRESENTATIE VAN QUERY RESULTATEN

Het resultaat van een get-opdracht kan worden gesorteerd. In het soreringscriterium mogen alleen direkte attributen voorkomen (d.w.z. geen attribuut-expressies en zonder gebruik van de its-constructie). De volgende query is een eenvoudig voorbeeld:

```
extend afdeling with staf =
        count werknemer per afdeling.
get afdeling its naam, staf per staf.
```

Na **per** wordt het sorteringscriterium opgenomen. In dit voorbeeld worden door **staf** de instances gesorteerd op oplopend staf aantal. Bij **-staf** worden de instances in aflopende volgorde weergegeven. Het sorteringscriterium mag uit meerdere elementen (die onderling

gescheiden worden door een komma) bestaan. De soort sortering (numeriek of alfanumeriek wordt door Xplain DBMS zelf afgeleid).

Indien men bij de presentatie van de query-resultaten toelichtingstekst wil zien kan men deze tekst tussen dubbele quotes opnemen achter de ’**its**’. Bijvoorbeeld:

get artikel **its** "naam=", beschrijving.

Het resultaat is een tabel met identificaties, herhaalde keren de tekst ’naam=’ gevolgd door de beschrijving van een artikel:

```
artikel       beschrijving
------------------------------
     1  naam=  radio
     2  naam=  kussen
     3  naam=  stoel
     :    :      :
```

Op soortgelijke wijze kan men zo een vertikaal streepje1 ’|’ tussen en achter resultaten verkrijgen.

get artikel **its** "| naam=", beschrijving, "|".

Om voor de identificaties een tekst te zetten moet men het volgende specificeren:

get "artikel-ID" artikel **its** "| naam=", beschrijving, "|".

Het resultaat is:

```
              artikel            beschrijving
-------------------------------------------------------
artikel-ID    art01     | naam=   artikel 1          |
artikel-ID    art02     | naam=   artikel 2          |
     :       :               :         :
```

Soms is alleen een lijst van identificaties gewenst. Bijvoorbeeld in een studentenadministratie waar het studentnummer als identificatie is gebruikt. Een overzicht van alleen identificaties (studentnummers) wordt verkregen door een lege string achter de ’**its**’ te plaatsen:

get student **its** "".

III.14. NEWLINE-OPDRACHT

Door de opdracht **newline** kan men in de uitvoer regelovergangen toevoegen aan het resultaat. Als deze opdracht in de attribuutlijst van een get-opdracht wordt geplaatst wordt de volgende attribuutwaarde op een nieuwe regel geplaatst. Plaatsing tussen twee get-opdrachten komt er

een extra lege regel tussen de twee lijsten.

Een voorbeeld ter illustratie:

```
get "code = " artikel its newline, "naam = ", beschrijving, newline.
newline.
get count artikel.
```

III.15. ECHO-OPDRACHT

De **echo**-opdracht kan gebruikt worden om tekst voor, tussen of na manipulatie-opdrachten te laten tonen op scherm. Hiermee kan men zorgen dat de gebruiker feed-back krijgt over de voortgang van de verwerking van de query. Dit kan wenselijk zijn bij grote of langdurende queries. Tevens kan men resultaten verduidelijken door ze te begeleiden met tekst. Bijvoorbeeld:

```
echo "Query wordt uitgevoerd, een ogenblik a.u.b.".
echo " Het aantal wordt bepaald.".
extend Atype with aantal =
        count Btype per Atype.
echo " Het totaal wordt bepaald.".
extend Atype with totprijs =
        total Btype its prijs per Atype.
echo "Het resultaat is:".
get Atype its aantal, totprijs.
```

III.16. COMMENTAAR IN QUERIES (#-OPDRACHT)

Er is een mogelijkheid om commentaar in de query op te nemen. Dit kan met het #-teken.

Alles op de regel **achter** het #-teken zal door de opdrachtenprocessor genegeerd worden. Als men commentaar wil specificeren die meer dan één regel omvat dan moet men aan het begin van iedere regel een #-teken zetten, bijvoorbeeld:

```
# Deze query bepaalt het geïnvesteerde vermogen
# van alle in voorraad zijnde artikelen.
get artikel its prijs * voorraad        # opvragen van geïnvesteerde vermogen
        where voorraad >= 1.  # voor ieder arikel in voorraad.
```

Commentaar kan, zoals in het bovenstaande voorbeeld getoond wordt, ook binnen delen van een opdracht gebruikt worden mits ze achter de delen op dezelfde regel staan.
Binnen karakterstrings, deze worden omringd door " (dubbele quote’s) wordt een # als een normaal karakter beschouwd.