Federatieve databases

FEDERATIEVE DATABASES

by

Simon J. Klaver

and

Chris F.M. Verberne

87/02

January 1987

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science of Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

editor: F.A.J. van Neerven

Simon J. Klaver
De Greefstraat 9a
5622 GJ Eindhoven
040 - 456543

Chris F. M. Verberne
Hoofdstraat 100
5757 AR Liessel
04934 - 1523

F E D E R A T I E V E D A T A B A S E S

I N H O U D

S A M E N V A T T I N G

De communicatie tussen geautomatiseerde informatiesystemen krijgt niet de aandacht die het verdient. Beschreven wordt hoe een van de belangrijkste bouwstenen van een informatiesysteem, het database-management-systeem, zo kan worden uitgebreid dat het automatisch uitwisselen van informatie tussen informatiesystemen eenvoudiger gerealiseerd kan worden. Het werk is een vervolg op het afstudeerverslag van de auteurs, aan de TH Eindhoven, geschreven onder begeleiding van Prof. Dr. K.M. van Hee en Dr. M. Voorhoeve.

1. Probleemstelling en oplossingsconcept.

We beschouwen een groep autonome, samenwerkende instanties waartussen een intensieve en omvangrijke informatieoverdracht plaatsheeft. Ook vandaag nog vindt deze informatieoverdracht nog op een handmatige manier plaats terwijl vele instanties de over te dragen informatie al in een eigen geautomatiseerd informatiesysteem hebben opgeslagen. Ook zijn grote delen van de informatiestromen van te voren te plannen omdat bepaalde uitgaande berichten vaak een vaste reactie zijn op bepaalde inkomende berichten.

Een voorbeeld van de hierboven geschetste situatie zijn de samenwerkende bedrijven en instanties in een transportketen. Hier is sprake van zeer omvangrijke overdracht van informatie waarvan grote delen afhankelijk zijn van elkaar. Deze informatiestromen komen handmatig tot stand hoewel een groot deel een routineaangelegenheid is. Er wordt relatief veel tijd besteed door mensen aan het opvragen van informatie die bij anderen in geautomatiseerde systemen aanwezig is. In Rotterdam wordt dit aantal geschat op 60 a 70 manjaren per jaar.

Verder gebeurt het soms dat een persoon die informatie moet verschaffen of krijgen een tijdlang niet bereikbaar is waardoor fikse vertragingen in de informatiestromen optreden. Het komt voor dat het informatieverkeer dat bij een transport hoort aanzienlijk later afgerond is dan het transport zelf, of dat het voor aanzienlijke vertragingen in het transport zorgt. Dit zorgt uiteraard voor veel extra kosten.

Het bovenstaande illustreert enigszins de omvang van het probleem. De informatiestromen in de haven zijn sterk afhankelijk van elkaar en daardoor zijn grote delen van te voren te plannen. Als voorbeeld voor deze samenhang kan een cargadoor in de haven dienen. Een cargadoor organiseert een gedeelte van het containertransport dat per schip gaat.

Hij krijgt de opdracht om voor de containers en het schip, en voor een reservering bij een terminal te zorgen, alwaar de containers in het schip geladen worden. Als de cargadoor zo'n opdracht krijgt kan hij automatisch de boekingen voor de containers, het schip en de reservering bij de terminal doen. Als deze boekingen bevestigd zijn dan kan automatisch de opdracht aan de cargadoor bij zijn opdrachtgever worden bevestigd.

Hieruit zien we dat het informatieverkeer een vast patroon heeft en dus te plannen is. In het hierboven geschetste voorbeeld zou in principe alles geautomatiseerd kunnen worden behalve in het geval dat er iets mis gaat, bijvoorbeeld als er een boeking geweigerd wordt.

We hebben ons tot taak gesteld het automatiseren van het informatieverkeer als het bovenstaande te vereenvoudigen. Dit doen we

door het uitbreiden van de functies van een van de belangrijkste bouwstenen van een informatiesysteem: het databasemanagement systeem.

Hiertoe willen we een DBMS uitbreiden met communicatiemogelijkheden op tupelniveau met andere gelijkwaardige systemen. Bovendien voegen we een programmeer-mechanisme toe dat de gebruiker in staat stelt een rijtje van transacties op de database (of een verzameling te versturen berichten) te definiëren die door het systeem worden uitgevoerd op het moment dat aan een (ook door de gebruiker ingevoerde) voorwaarde, is voldaan. Met zo'n mechanisme wordt het mogelijk zonder menselijke tussenkomst berichten te doen uitgaan als reactie op een ingaand bericht.

Communicatie tussen databases zal zich natuurlijk beperken tot die delen waarvoor lees- of schrijfrechten zijn toegewezen. Een verzoek van een ander systeem aan een database om data stellen we ons voor als een query in een geschikte vraagtaal die over het netwerk wordt verstuurd dat alle informatiesystemen van de betrokken deelnemers met elkaar verbindt.

Deze opzet brengt met zich mee dat ieder deelnemend systeem de beschikking zal hebben over dezelfde vraag/manipulatietaal ofwel een en dezelfde standaard database-taal.

Het belangrijkste kenmerk van dit systeem is dat alle systemen autonoom blijven. Er is geen big-brother die alle communicatie regelt. Iedereen kan zelf bepalen welke rechten m.b.t. het bekijken of veranderen van zijn database hij aan anderen toestaat. Het verschil met een gedistribueerde database is dat zich in ons netwerk een aantal op fysiek en logisch niveau te scheiden, ofwel federatieve, databases bevinden die af en toe informatie uitwisselen, terwijl een gedistribueerde database in feite slechts één database is waarvan elke deelnemer aan het netwerk een deel in bewaring heeft. Een deelnemer kan aan andere systemen opvragen welke tabellen voor hem toegankelijk zijn.

In de rest van dit verhaal geven we een beschrijving van zo'n uitgebreid databasemanagement-systeem. De basis voor onze beschrijving vormt een relationeel DBMS, dat bij iedere deelnemer aanwezig moet zijn, met de vraag/manipulatietaal SQL. In [1] is een syntax van SQL opgenomen en in [2] en [3] wordt dieper ingegaan op de mogelijkheden van SQL.

Het DBMS dat we gaan beschrijven zullen we een FEDERATIEVE DATABASE noemen. Een voorgeprogrammeerd rijtje transacties of berichten samen met hun activerings voorwaarde zullen we een RULE noemen.

2. Systeembeschrijving:

2.1. In-box, Mail-box, Out-box

Een federatieve database uitgebreid met een rule mechanisme kunnen we als volgt karakteriseren:

1) het systeem heeft tenminste de functies van een relationeel database management systeem.

2) het systeem is via een netwerk met een aantal soortgelijke systemen verbonden.

3) het systeem kan opdrachten (in de database-taal) ter uitvoering naar andere deelnemers sturen.

4) het systeem kan (met enige restricties) tabellen, die zich bij andere systemen bevinden, gebruiken (oftewel data opvragen bij anderen).

5) het systeem is programmeerbaar: de gebruiker kan een van te voren definieerd rijtje opdrachten in de database taal, laten uitvoeren op het moment dat aan een door de gebruiker bepaalde voorwaarde wordt voldaan.

In dit gedeelte schetsen we die delen van ons DBMS die specifiek zijn voor ons systeem.

Ter wille van de communicatie met andere soortgelijke systemen zijn er in ons systeem drie zogenaamde brievenbussen aanwezig. Deze brievenbussen fungeren als buffers van opdrachten tussen het systeem en het netwerk.

a) MAIL-BOX · voor binnengekomen vrije-tekst-berichten (een soort electronic mail functie voor de gebruikers).

b) JOB-BOX voor de door het systeem nog te behandelen opdrachten

c) OUT-BOX voor alle uitgaande berichten en opdrachten voor andere systemen

Deze brievenbussen zijn geordende rijen van opdrachten en berichten die ieder in één tabel zijn opgeslagen. Berichten in de OUT-BOX zijn voorzien van de naam van de bestemming van het bericht. Via datacommunicatieprogrammatuur wordt ervoor gezorgd dat deze berichten automatisch uit de OUT-BOX verdwijnen en voorzien van de naam van de afzender in de MAIL of JOB-box van de ontvanger terecht komen. Dit stuk programmatuur plus hardware noemen we voortaan het netwerk.

Het systeem heeft twee prioriteiten voor opdrachten. De laagste voor opdrachten in de JOB-BOX en de hoogste voor opdrachten die afkomen van een interactieve gebruiker. Als er geen opdracht van een interactieve gebruiker (of gebruikersprogramma) worden aangeboden worden de opdrachten uit de JOB-BOX behandeld.

Een RULE zal intern in het systeem worden gekarakteriseerd door een rulenaam, de naam van een view (een afkorting van een select-statement), de naam van een job (een logic unit of work) en een actiefveld (een boolean). Op het moment dat door een toestandsverandering van de database de inhoud van een view van een geactiveerde rule van leeg naar niet leeg gaat wordt de bijbehorende job ter uitvoering vooraan in de JOB-BOX gezet.

Of een rule geactiveerd is of niet wordt in het actiefveld bijgehouden. Vóór uitvoering van de job wordt het actiefveld op false gezet.

2.2. De opdrachten:

De Database taal die we hier voorstellen is een uitbreiding van SQL. We veronderstellen het gebruik hiervan bekend. Hoewel er vele aanmerkingen te maken zijn op SQL hebben we het vanwege de bekendheid en de weinige alternatieven toch als basis voor onze taal gekozen.

De eerste twee voorbeelden geven aan hoe we SQL willen uitbreiden zodat queries op een externe database kunnen worden uitgevoerd.

De voorbeelden maken gebruik van de tabel 'Adres' die aanwezig is bij deelnemer 'Verberne' en de tabel 'Klanten' die aanwezig is bij deelnemer 'Klaver'.

V1a) Klaver stuurt naar Verberne een aantal adressen.

```
Insert Into Adres (Nummer, NAW)
At     Verberne
Select Nummer, NAW
From   Klanten
Where  Nummer Between 'A' and 'Bzzz'
/
```

V1b) Verberne vraagt aan Klaver dezelfde adressen.

```
Insert Into Adres (Nummer, NAW)
Select Nummer, NAW
From   Klanten
At     Klaver
Where  Nummer Between 'A' and 'Bzzz'
/
```

Beide voorbeelden hebben het effect dat een aantal adresgegevens die zich bij Klaver in tabel Klanten bevinden bij Verberne in de tabel Adres worden gestopt.

Deze twee voorbeelden maken gebruik van het nieuwe reserved word At, dat aangeeft bij welk systeem zich een tabel of een aantal tabellen bevinden.

Om het implementeren zo eenvoudig mogelijk te houden sluiten we de mogelijkheid uit om in één SELECT-statement verschillende tabellen te gebruiken die zich bij meerdere deelnemers bevinden. (Ook het algoritme dat het optimale pad door de database bepaalt om een query uit te rekenen is in zo'n geval onbruikbaar).

Het is dus niet toegestaan om het volgende op te schrijven:

```
Select *
From   Reizen     At Piet,
       Containers At Jan
Where  Reizen.nr = Containers.nr
/
```

SELECT-statements van deze vorm kunnen altijd met de door ons voorgestelde beperkte manier worden uitgerekend door de tabellen die nodig zijn eerst naar het eigen systeem te copieren.

Voor electronic-mail heeft het systeem de Mail-opdracht:

```
V2) Mail Jansen
    "Geachte collega,

     bij deze ontvangt U de omzet cijfers van het laatste kwartaal:

    ", Select * From Omzet Where Weeknr Between 1 and 13
    /
```

Met een mail-opdracht kunnen stukken tekst met, daarin geintegreerd, delen uit de database naar andere systemen gestuurd worden.

De volgende voorbeelden behandelen het beheer van rules. In het voorbeeld wordt een rule geactiveerd die automatisch het salaris van de werknemers verhoogt zodra de jaarwinst boven de miljoen komt.

```
V3a) Create Job Verhoogsalaris As
     [[ Update Pers
        Set    Sal = Sal * 1.03
        Where  Systime - Indienst > 25*365
      ;
        Update Pers
        Set    Sal = Sal * 1.02
        Where  Systime - Indienst <= 25*365
     ]]

V3b) Create View Winstbovenmiljoen          -- dummy is een tabel
     As Select *                            -- met tenminste een rij
        From   Dummy
        Where  1000000 < Select Count (Winst)
                         From   Omzet

V3c) Create Rule Veelwinst
     When Winstbovenmiljoen
     Do Verhoogsalaris

V3d) Activate Rule Veelwinst
```

2.3. De syntax van de databasetaal opgeschreven in BNF

Terminal symbolen staan tussen dubbele aanhalingstekens. Tussen rechte haken zetten van een aantal symbolen betekent 0 of 1 keer herhalen. Tussen accolades zetten betekent 0 of meer keer herhalen. Een verticale streep betekent dat je moet kiezen uit de symbolen voor en de symbolen na de streep. Dezelfde BNF-syntax is te vinden in [4].

```
01 Logic-Unit-of-Work  ::= Database-statement {";" Database-statement} "/"

02 Database-statement  ::= SQL-statement
                         | Mail-statement
                         | Rule-statement
                         | Job-statement

03 SQL-statement       ::= Select-statement
                         | Insert-statement
                         | Mr-insert-statement
                         | Update-statement
                         | Create-statement
                         | Delete-statement

04 Select-statement    ::= Select-part From-part [Where-part]
                           [Group-part] [Order-part]

05 Insert-statement    ::= "INSERT INTO" Table-part Value-part

06 Mr-insert-statement ::= "MRINSERT INTO" Table-part Constant-part

07 Update_statement    ::= "UPDATE" Table-part Set-part

08 Delete-statement    ::= "DELETE FROM" Table-part [Where-part]

09 Create-statement    ::= Create_Job_statement
                         | Create_Rule_statement
                         | Create_View_statement
                         | Create_Table_statement

10 Create_Job_statement
                       ::= "CREATE JOB" Jobname "AS"
                           "[[" Logic-Unit-of-Work "]]"

11 Create_Rule_statement
                       ::= "CREATE RULE" Rulename
                           "WHEN" Viewname Job-statement

12 Create_View_statement
                       ::= "CREATE VIEW" Viewname "AS" Select-statement
13 Create_Table_statement
                       ::= "CREATE TABLE" Tablename
                           "(" Attrib_decl {"," Attrib_decl } ")"

14 Mail-statement      ::= "MAIL" Deelnemerid  Mail-item {"," Mail-item}
```

```
15 Rule-statement       ::= "ACTIVATE RULE" Rulename
                          | "DEACTIVATE RULE" Rulename

16 Job-statement        ::= "DO" Jobname

17 From-part            ::= Tablename {"," Tablename} [At-part]

18 Mail-item            ::= '"' stringconstante '"' | Select-statement

19 Table-part           ::= Tablename [Attributes] [At-part]

20 Attributes           ::= "(" Columnname {"," Columnname} ")"

21 At-part              ::= "AT" Deelnemerid

22 Constant-part        ::= "(" Constants {"," Constants} ")"

23 Constants            ::= "(" Constant   {"," Constant  } ")"
```

De nonterminals Tablename, Viewname, Jobname, Deelnemerid en Constant zijn willekeurige strings. De nonterminals Select-part, Where-part, Valuepart, Set-part en Order-part worden in de SQL-syntax uitgewerkt. Mr-insert (Multirow-Insert) wordt in 2.4 uitgelegd.

Buiten deze syntax om definieren we een pre-processor mechanisme dat het mogelijk maakt om stukken van een database statement pas tijdens de uitvoering ervan definitief in te vullen. Dit is vooral handig bij opdrachten die door een rule worden uitgevoerd. Zo'n variabel stuk tekst uit een database-statement noemen we een parameter. In BNF:

```
parameter       ::=   "#" parameternaam ":[" select-statement "]#"
                    | "#" parameternaam ":[USER]#"
                    | "#" parameternaam "#"

parameternaam   ::=   Mail-item
```

Voordat een database opdracht wordt verwerkt wordt de tekst van de parameter vervangen door zijn waarde. De waarde van een parameter is in het eerste geval de waarde van het eerste attribuut van het eerste tupel uit het resultaat van het select-statement.

In het tweede geval wordt de gebruiker om een vervangende tekst gevraagd voor de parameter. In het laatste geval wordt als waarde de parameter met gelijkluidende naam uit dezelfde database-opdracht genomen.

Tijdens het invullen van een waarde voor een parameter krijgt de gebruiker de parameter naam te zien. In geval van een gebruikersparameter kan dit het resultaat van een select-statement zijn.

2.4. Semantiek

Een logic-unit-of-work bestaat uit een aantal database-statements die elk tot een van de volgende categorien behoren:

Mail-statement: Het mail-statement vervangt alle select-statements uit de mail-items door hun waarde en stuurt daarna alle items aan elkaar geplakt naar de mail-box van de betreffende deelnemer.

Rule-statement: Dit dient om rules op actief of non-actief te kunnen zetten. Een rule die actief is en waarvan wordt gedetecteerd dat de voorwaarde true is geworden, is zet zichzelf op non-actief en voert dan de rij database-opdrachten uit die in de genoemde job staan.

Job-statement: Hiermee kan een van te voren van een naam voorziene logic-unit-of-work (d.m.v. CREATE JOB) met een statement worden uitgevoerd.

SQL-statement: Dit dient om standaard SQL-statements uit te voeren met als uitbreidingen:

- Een insert-statement (MR-INSERT), met nagenoeg dezelfde syntax als de gewone INSERT, dat het mogelijk maakt om in één statement meerdere constante tupels aan een tabel toe te voegen.
- Het create statement waarmee ook JOB's en RULE's kunnen worden gedefinieerd.
- Tabellen hoeven niet altijd op het eigen systeem aanwezig te zijn. De gebruiker kan met een At-part aangeven bij welke deelnemer in het netwerk de betreffende tabel of tabellen zich bevinden.

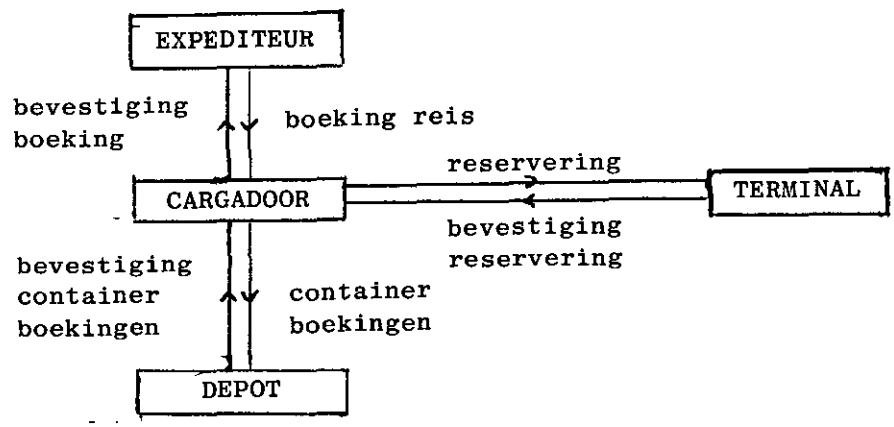
3. Toepassing op transportprocessen

3.1 Probleemschets

We beschouwen een transport van beladen containers die per vrachtauto, van de plaats van herkomst, naar een terminal in de haven worden gebracht. Daar worden de containers overgebracht in een schip dat deze containers naar de haven van bestemming brengt.

De cargadoor speelt hierin een centrale rol. Hij krijgt van de expediteur, dit is hier de organisator van het transport, de opdracht om voor containers te zorgen via een containerdepot en om voor een reservering bij de terminal te zorgen waar de containers in het schip worden geplaatst. Een container depot is hier een containerverhuurbedrijf.

We werken hier bij wijze van voorbeeld het systeem van de cargadoor uit. De berichtenflow is fictief en dient alleen ter illustratie van de kracht het gebruik van onze oplossing. We kunnen de berichtenstroom schematisch als volgt weergeven:



Bij de cargadoor start de procesgang doordat de reisboeking van de expediteur in zijn systeem binnenkomt. Een reis omvat in onze context het vervoer van een groep containers van verschillende herkomst naar één bestemmingshaven. De cargadoor vult voor nieuw binnengekomen reizen zijn referentie-code in en stuurt boekingen voor bijbehorende containers naar het containerdepot.

Na bevestiging van de containerboekingen door het depot wordt een reservering naar de terminal gestuurd. Als de terminal met deze reservering akkoord gaat dan stuurt de cargadoor een bevestiging voor de reisboeking naar de expediteur terug. Als de terminal of het depot niet positief hebben gereageerd m.b.t. de reis dan moet deze reis aan de cargadoor worden gemeld met de mededeling dat de terminal of het depot niet akkoord ging.

3.2 Het database model

Hier werken we het databasemodel van de cargadoor uit. Reizen worden in de tabel 'Reis' opgenomen. Als de expediteur hier een tupel invult dan wordt dat als een reisboeking beschouwd (De sleutel is telkens onderstreept).

```
reis   :=  { ( reisnr              : number)
           , ( id_expediteur       : char (9))
           , ( id_depot            : char (9))
           , ( id_terminal         : char (9))
           , ( loshaven            : char (9))
           , ( status              : char (6))
           , ( ref_code            : char (9))
           }
```

In de tabel reis staan de identificaties van de terminal en het depot bij iedere reis opgenomen. Deze dienen door de cargadoor voor iedere nieuw bijgekomen reis te worden ingevuld, alsmede de referentie-code van de cargadoor voor die reis. Verder is bij iedere reis een loshaven opgenomen die aan de cargadoor bekend moet zijn i.v.m. de keuze van de terminal. Het status-veld bij een reis is een element uit de verzameling {'B', 'A', 'Bk', 'BD', 'BT'}. De betekenissen hiervan zijn resp. bevestigd, afgewezen, reis moet i.v.m. niet akkoord gaan van depot of terminal nog eens worden bekeken, boeking voor containers is naar depot verstuurd, reservering terminal is verstuurd.

De containers die bij een reis horen staan in de volgende tabel:

```
container  :=  { ( contvolgnr      : number)
               , ( reisnr          : number)
               , ( uit_afl_dep     : date)
               , ( verw_aank_term  : date)
               , ( omschr          : char (50))
               }
```

Een container wordt uniek geïdentificeerd door het contvolgnr. Dit wordt, evenals elk veld in deze tabel, ingevuld door de expediteur d.m.v. de reisboeking.

Om de bevestiging van het depot in de database op te nemen wordt een aparte tabel gemaakt. Apart omdat het depot niet hoeft te weten wat voor gegevens de cargadoor allemaal over de containers bijhoudt. Deze tabel heet dep_stat.

```
dep_stat  :=  { ( contvolgnr       : number)
              , ( status_dep       : char (6))
              }
```

De status is een element uit {'B', 'NB', 'NOTB'}. Deze afkortingen staan resp. voor beschikbaar, niet beschikbaar en niet op tijd beschikbaar.

Evenzo wordt in een aparte tabel, term_stat, geregistreerd of de terminal met een reservering akkoord gaat.

```
term_stat :=   { ( reisnr          : number)
               , ( status_term     : char (6))
               }
```

Het status veld kan de waarde 'OK' (akkoord) of 'NOT OK' (niet akkoord) aannemen.

3.3 Uitwerking rules cargadoor

Hier worden de 5 rules beschreven die de werking van het locale systeem van de cargadoor bepalen.

rule 1: Deze rule vraagt aan de cargadoor de referentiecode, het adres van het depot en de terminal, van alle reizen die nieuw zijn bijgekomen in de Reis-tabel, onder vermelding van de loshavens. Daarna worden deze gegevens ingevuld en activeert de rule zichzelf. Het maken van een rule gebeurt als volgt:

```
create view view1
as      select reisnr
        from   reis
        where  status is NULL  and  ref_code is NULL
        and    id_dep is NULL  and  id_terminal is NULL
;
create job job1
as      update reis
        set ref_code = #"referentiecode":[user]#
          , id_terminal  = #"Geef terminal_id voor loshaven: "
                         , select loshaven from reis
                           where reisnr in
                           (select reisnr from view1
                            where rownum = 1)
                         :[user]
                         #
          , id_dep   = #"depot-id":[user]#
         where reisnr in
              (select reisnr from view1
               where rownum = 1)
         ;
         activate rule rule1
         /
;
create rule rule1
when    view1
do      job1
/
```

Voor de exacte semantiek van het hier gebruikte SQL-dialect verwijzen we naar [5].

rule2: Deze rule stuurt containerboekingen naar het depot voor reizen waarvoor dit nog niet is gebeurd en die al wel door rule1 zijn behandeld (identificatie depot en terminal alsmede de referentiecode zijn ingevuld).

```
create view view2
as      select reisnr
        from   reis t
        where  status is NULL
        and    ref_code is not NULL
        and    id_dep is not NULL
        and    id_terminal is not NULL
        and    0 != (select count (*)
                     from   container
                     where  t.reisnr = reisnr)
;
create job job2
as      insert into boeking
        at #:[select id_dep
              from   reis
              where  reisnr in (select reisnr from view2)
              and    rownum = 1
           ] #
        select ref_code, contvolgnr, to_char (uit_afl_dep, 'DD/MM/YY')
        from   reis, container
        where  reis.reisnr = container.reisnr
        and    reis.reisnr in (select reisnr
                               from   view2
                               where  rownum = 1)
        ;
        update reis
        set    status = 'BD'
        where  reisnr in (select reisnr
                          from   view2
                          where  rownum = 1)
        ;
        activate rule rule2
        /
;
create rule rule2
when   view2 do job2
/
```

rule 3: Deze rule zorgt ervoor dat reserveringen naar de terminal worden gestuurd voor reizen waarvan de containerboekingen bevestigd zijn en waarvoor nog geen reservering gedaan is. De status van deze reizen wordt 'BT' gemaakt en rule3 wordt weer geactiveerd.

rule4: Deze rule stuurt bevestigingen naar de expediteur voor reizen die zowel door het depot als door de terminal zijn bevestigd en waarvoor nog geen bevestiging is verstuurd. De status van deze reizen wordt dan 'B' en rule4 wordt geactiveerd.

rule5: Deze rule meldt de reizen aan de cargadoor waarop door het depot of de terminal negatief gereageerd is en die nog niet zijn gemeld. De status wordt dan 'Bk' gemaakt en rule5 wordt geactiveerd.

Het maken van rule3 t/m rule5 gaat analoog aan rule1 en rule2.

5. Conclusies

In de eerste helft van 1986 hebben de auteurs tijdens hun afstudeerperiode bij de vakgroep Informatica van de T.H. Eindhoven, onder leiding van Professor K.M. van Hee en Dr. M. Voorhoeve een prototype gebouwd waarvan de specificaties voor een belangrijk deel overeenkwamen met het systeem uit de voorgaande tekst.

Uit de ervaringen die we tijdens deze periode hebben opgedaan durven we te concluderen dat de oplossing die we hier aangeven voor het versimpelen van het berichten verkeer tussen informatiesystemen, en databases in het bijzonder, de volgende meerwaarde oplevert:

- Een deel van de berichten kan automatisch verstuurd worden.
- Een deel van de binnenkomende berichten in een systeem kan dankzij rules automatisch worden verwerkt.
- Gegevens opvragen die niet op het eigen systeem aanwezig zijn kan nu, mits er toe gerechtigd, gebeuren zonder dat er aan de andere kant iemand aanwezig hoeft te zijn, bijvoorbeeld 's nachts.
- Het implementeren van een koppeling tussen twee berichten is sterk vereenvoudigd en in korte tijd te leren. Bij het implementeren van zo'n koppeling is minder vaak een programmeur nodig. Het systeem is programmeerbaar door (geoefende) eindgebruikers i.p.v. professionele programmeurs.

De eerste drie punten konden we met ons prototype helaas niet echt overtuigend demonstreren vanwege de grote beperkingen van de voor het systeem gebruikte PC, zoals het ontbreken van geschikte netwerk faciliteiten.

Het laatste punt wordt meteen duidelijk bij het zelf uitwerken van een voorbeeld met een paar rules. Zodra duidelijk is wát gemaakt moet worden is meteen duidelijk hóe het gemaakt moet worden. En zo hoort het ook!

[1] "Geschakelde Databases", S. J. Klaver en C. F. M. Verberne, afstudeerverslag Technische Hogeschool Eindhoven, mei 1986.

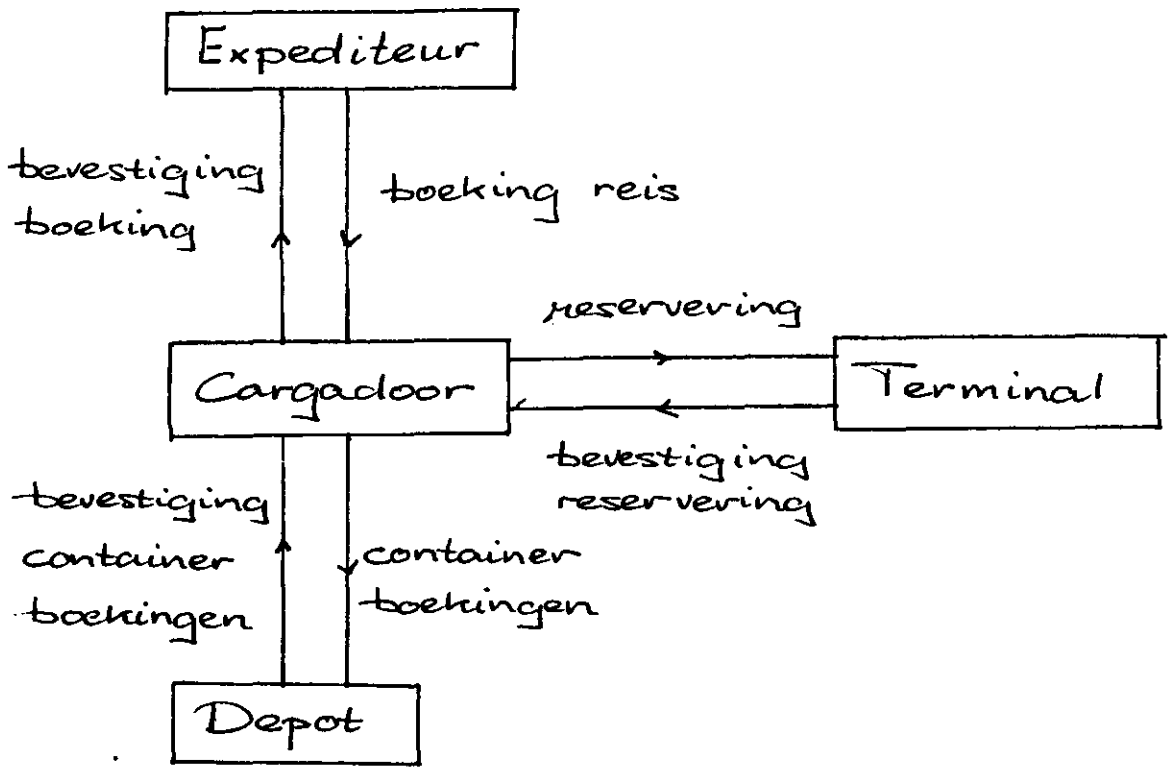
[2] "SQL in de praktijk", Eilers e. a., Academic Service 1985.

[3] "Onderzoek naar de mogelijkheden en performance van Oracle", E.H. Reitsma, THE 1984.

[4] "Modula-2", N. Wirth, Springer Verlag 1985.

[5] "UFI/SQL version 4.1.4 reference manual", Oracle Corp. 1984.

Kladpapier


Expediteur
bevestiging boeking
boeking reis
reservering
Cargadoor
Terminal
bevestiging reservering
bevestiging container boekingen
container boekingen
Depot