Verwijderprocedure ouder beperkt

TUE-RC 85597

Verwijderprocedure ouder beperkt

Computing Centre Note 53

Stageverslag door:
Benny Pinontoan
Faculteit Informatica

Begeleiders : ing. C.N.M. du Bois
T.H. Bruning

Augustus 1991,
Technische Universiteit Eindhoven, Rekencentrum.

VOORWOORD

Deze stage wordt gedaan binnen het kader van mijn studie aan de Faculteit Informatica, Technische Universiteit Eindhoven. Deze stage is verricht bij het Rekencentrum, een instantie binnen TUE die verantwoordelijk is voor het verschaffen van automatiseringsfaciliteiten en diensten aan de Universiteit en ook aan externe gebruikers.
De keuze voor dit onderwerp is een goede steun geweest aan mijn verdere studie vooral in het gebied van database systemen. Hiervoor wil ik mijn stagebegeleiders, de heer du Bois en de heer Bruning, bedanken voor het beschikbaar stellen van dit onderwerp en hun begeleiding tijdens mijn stageperiode.
Op deze plaats wil ik ook de heer Korver en de heer Verkissen en mijn medestagiair Hans Scheepers bedanken die heel erg behulpzaam zijn geweest voor het verhelpen van de problemen tijdens de stage. Verder mijn dank aan andere medestagiairs voor hun hulp en gezelschap.

INHOUDSOPGAVE

1. INLEIDING

1.1 De Aanleiding tot de Opdracht

Deze stageopdracht sluit aan een afstudeeropdracht [HELD].
Bij die afstudeeropdracht worden procedures ontworpen voor het generatieproces die de benodigde procedures voor het in stand houden van de referentiële integriteit in een te genereren applicatie kunnen genereren.
Gemakshalve worden verder in dit verslag de eerste soort procedures generatieprocedures genoemd, en de laatste soort procedures als integriteitsprocedures genoemd.
Er zijn verschillende integriteitsprocedures mogelijk.
Deze worden opgesomd in § 4.1.

Een paar van de generatieprocedures zijn al bij die afstudeeropdracht geïmplementeerd. Er zijn dus nog enige generatieprocedures overgebleven die geïmplementeerd moeten worden.

Dit was de aanleiding geweest voor deze stageopdracht, om één van de overgeblevende generatieprocedures te implementeren.

De precieze formulering van de opdracht vindt plaats in de volgende paragraaf.

1.2 Opdracht Formulering

Gezien de zeer beperkte tijd voor de uitvoering van de opdracht, wordt in eerste instantie maar één van de overgeblevende generatieprocedures uitgekozen.

De keuze wordt zodanig gemaakt dat deze één van de meest essentiële procedures is en dat deze realiseerbaar is binnen de gegeven tijd.
De keuze valt op "de verwijderprocedure voor OUDER-BEPERKT".
Voor het begrip OUDER zie [HELD], blz.25 en voor het begrip BEPERKT zie § 4.1.

De opdracht zelf is te splitsen in de volgende drie onderdelen:

a). Bekijk hoe de referentiële integriteit in (SQL*Forms) applicatie gehandhaafd kan blijven bij een verwijdering, die als eigenschap BEPERKT heeft, in de tabel die optreedt als OUDER-tabel, en opstellen hoe de benodigde integriteitsprocedure er uit moet zien.

b). Ontwerpen van de generatieprocedure die de bij a) gevonden integriteitsprocedure genereert.

c). Implementeren en testen de ontworpen generatieprocedure in het generatorproces.

Samengevat kan de opdracht als volgt geformuleerd worden:

> "Ontwerp procedure voor het generatieproces die de verwijderprocedure OUDER-BEPERKT in een te genereren (SQL*Forms) applicatie kan genereren.
> Implementeer en test deze."

1.3 Uitgangspunten

De uitgangspunten die in [HELD], § 1.7 aangenomen zijn gelden hier ook.

1.4 Planning voor de uitvoering van de opdracht

De geplande tijdsduur voor deze stage is 30 middagen.
Bij de aanvang van de stage is er een planning opgesteld hoe het verloop van de stage ongeveer zal zijn.
Deze wordt opgedeeld in blokken van 5 middagen en ziet als volgt uit:

De eerste 5 middagen:	Bestuderen van de opdracht en leren omgaan met de apparatuur (in dit geval IBM 4381).
De tweede 5 middagen:	Bestuderen van ORACLE-SQL*PLUS en oefenen met deze.
De derde 5 middagen :	Bestuderen van ORACLE-SQL*Forms en oefenen met deze.
De vierde 5 middagen:	Bestuderen van het generatieproces en Opstellen van de verwijderprocedure OUDER-BEPERKT.
De vijfde 5 middagen:	Implementeren en testen van de generatie-procedure.
De zesde 5 middagen :	Eventuele voortzetting van testen en Verslag maken.

2. GLOBALE BESCHRIJVING VAN ORACLE

2.1 Wat is ORACLE ?

ORACLE is een modern Relationeel Database Management Systeem (RDBMS). In het algemeen treedt deze op als een doorzichtige interface tussen de fysieke opslag en de logische presentatie van de data.
ORACLE is een modulair systeem dat is opgebouwd uit de Oracle database en diverse functionele programma's. Deze componenten kunnen worden beschouwd als van elkaar onafhankelijke instrumenten waarmee men kan gebruiken voor :

- het beschrijven van een database;
- het zoeken van een database;
- het invoegen, bewerken, en uitwissen van gegevens;
- het invoeren en uitvoeren van gegevens;
- het wijzigen van de structuur van een database;
- het ontoegankelijk maken van gegevens voor onbevoegden;
- het communiceren binnen netwerken;

ORACLE bestaat onder andere uit de volgende modules:

- EASY*SQL : Een snelle en eenvoudige manier om toegang te leren krijgen en vervolgens ermee te leren werken.
- SQL*PLUS : Een belangrijkste interface die direct toegang geeft tot de RDBMS ORACLE. Met deze kan men een bestand zoeken en data definiëren, manipuleren en controleren. Vooral voor een Database Administrator, DBA, is deze een prima hulpmiddel.
- SQL*Forms : Een instrument waarmee op een formulier-gestuurde manier de database geraadpleegd en onderhouden kan worden.
- SQL*Reportwriter : Een menu-gestuurde format instrument dat gebruik maakt van SQL bij de productie van professionele rapporten.
- SQL*Calc : Een spreadsheet-interface van ORACLE-data.
- SQL*Graph : Een grafish hulpmiddel dat gegevens uit de SQL database omzet in visuele presentatie.
- SQL*QMX : Een optioneel programma voor het schrijven van rapporten waarmee men kan opzoeken aan de hand van voorbeelden.

Bij de uitvoering van deze stageopdracht zijn alleen SQL*Plus en SQL*Forms van belang. Deze worden uitgebreid beschreven in de volgende paragraaf.

2.2 Beschrijving van de relevante ORACLE-modules

SQL*Plus

Van alle module's die het Oracle-systeem kent, geeft SQL*Plus de meest direct toegang tot de data.
SQL*Plus, net als de andere Oracle-instrumenten, gebaseerd op de ANSI-Standaard SQL. De meeste commando's in SQL*Plus zijn ook geschreven met behulp van ANSI standaard commando's.
Deze commando's zijn tevens overdraagbaar naar elk op SQL gebaseerd systeem. Daarnaast heeft SQL*Plus ook aanvullende commando's die specifiek zijn voor SQL*Plus.
SQL*Plus biedt de mogelijkheden om een database te construeren, d.m.v. tabel te ontwerpen, de bijbehorende velden te definiëren, en data in te voeren. Met SQL*Plus-statements kan men database bevragen, views maken, en database onderhouden.

De mogelijkheden die SQL*Plus aanbiedt zijn echter nog niet voldoende om een echte applicatie te bouwen.
Met SQL*Plus zijn bijvoorbeeld zaken zoals repetitie en conditionele statements niet te realiseren.
Voor het realiseren van deze statements heeft ORACLE een ander alternatief.

SQL*Forms

Zowel de ANSI-Standaard SQL als de aanvullende SQL*Plus commando's zijn van groot belang bij het manipuleren van data.
Maar in de dagelijkse praktijk is een wat makkelijke manier om data in te voeren geen overbodige luxe.
Hiervoor heeft ORACLE een veelzijdig instrument voor het genereren van formulieren: SQL*Forms.

SQL*Forms is een vierde-generatie-taal-tool, formulier-gestuurd instrument waarmee het mogelijk is snel database applicaties te ontwikkelen.
Deze database applicaties hebben de volgende functies:

- Gegevens uit de database benaderen en manipuleren;
- Via schermformulieren en besturing daarvan (als userinterface) de database op een gebruikersvriendelijke manier benaderen;
- Consistentie-controles uitvoeren, zoals validiteit van de gegevens en het waarborgen van database constraints.

De database applicatie welke het betreffende schermformulier als userinterface heeft, kan in drie vormen optreden:

- Een tabelvorm die in het datadictionary-systeem van ORACLE is opgenomen (intern database formaat);
- Extern source formaat (INP-file);
- Extern executeerbaar formaat (FRM-file).

De ontwikkeling van een SQL*Forms-applicatie geschiedt in de volgende vier stappen:

- Ontwikkeling van een schermformulier;
  Een schermformulier kan meerdere blokken bevatten. Ieder blok correspondeert met één basis tabel, welke door de applicatie benaderd kan worden. Een blok kan meerdere velden bevatten. Deze velden corresponderen met de kolommen van de basis tabel.
- Aanpassing en uitbreiding van het schermformulier (bijvoorbeeld met extra velden);
  Deze wordt gedaan met behulp van de scherm-editor.
- Toevoegen van veld-specificaties en veld-validatie-criteria.
- Definiëren van triggers;
  Triggers zijn operaties, welke door het systeem uitgevoerd worden bij het optreden van een bepaalde gebeurtenis. Bijvoorbeeld bij het drukken op een functie-toets of bij het invoeren van waarden.
  Met behulp van deze triggers kan bijvoorbeeld de integriteit van de database of de veld-validiteit gehandhaafd blijven.

2.3 De mogelijkheid voor het genereren van SQL*Forms-applicaties

Zoals in vorige paragraaf vermeld, zijn er drie representaties van SQL*Forms-applicaties zijn: Tabellenrepresentatie, INP-file, en FRM-file.
De eerste twee representaties zijn wel te doorzien, terwijl FRM-file representatie niet direct te doorzien is.
De structuur en syntax van INP-file is makkelijk te achterhalen en te genereren. Een generator van deze file kan in hogere programeertalen geschreven worden. ORACLE zelf biedt te weinig mogelijkheden om een dergelijke file te genereren.
De eerste representatie, de tabellenrepresentatie, kan echter met een SQL*Forms-applicatie gemanipuleerd worden.
De applicatie-generator is dan wel te implementeren als SQL*Forms-applicatie.
De gegevens die nodig zijn voor de applicatie-generator zijn ook te manipuleren in SQL*Forms-applicatie.

3. BESCHRIJVING VAN HET GENERATIEPROCES

3.1. Voorwaarden voor het generatieproces

Zoals in § 2.3 vermeld wordt, bestaan er mogelijkheden voor het genereren van SQL*Forms-applicaties.
Om een applicatie te kunnen genereren moet deze aan een aantal eisen voldoen. Deze eisen worden om praktische redenen gesteld.

Als eerste wordt geëist dat bij de te genereren applicatie, per tabel, de sleutel éénduidig is. Hiermee wordt bedoeld dat er geen alternatieve sleutels mogen voorkomen. Deze eis wordt opgelegd om het generatieproces niet ingewikkelder te maken.

Als tweede wordt geëist dat er in het datamodel alleen maar 1-op-N-relaties voorkomen. Deze eis komt overeen met uitgangspunt 7 (zie [HELD], blz 17-18).

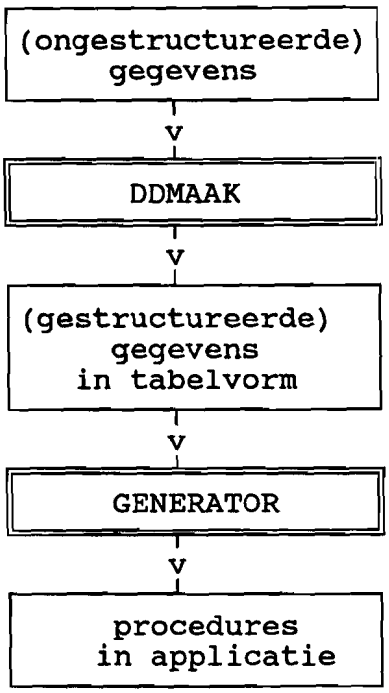
De derde eis heeft betrekking met de naamgeving van de tabellen; De naamgeving moet zodanig worden gekozen dat deze niet de naam "ZZZZZZZZZZ" heeft. Deze naam wordt namelijk als de afsluiter gebruikt.

3.2 De Structuur van het Generatieproces

De applicatiegenerator kan alleen maar functioneren indien er een aantal gegevens over de te genereren applicatie bekend zijn. Zo moet bijvoorbeeld eerst vastgelegd worden over welke tabellen het gaat en welke bijbehorende velden van de tabellen de relatie bewerkstelligen.
Er zijn dan ook twee subprocessen te onderscheiden binnen het generatieproces, namelijk:

- Het vastleggen van de benodigde gegevens;
- Het genereren van de gewenste procedures.

De eerste functie wordt verder met "DDMAAK" en de tweede met "GENERATOR" aangeduid.
De DDMAAK verwerkt de (ongestructureerde) gegevens, dat zijn gegevens over de applicatie zoals die door de gebruiker van het generatieproces opgesteld zijn, tot (gestructureerde) gegevens in tabelvorm. De GENERATOR gebruikt de gegevens in deze vorm voor het genereren van de integriteitsprocedures.
Het generatieproces kan dan in het volgende diagramflow weergegeven:



figuur 1: Het diagramflow van het generatieproces.

3.3 De Functionaliteit en Structuur van DDMAAK

Het proces "Het vastleggen van de benodigde gegevens", de DDMAAK, dient ervoor te zorgen dat de gegevens die nodig zijn voor het generatieproces in tabelvorm worden vastgelegd. Dit kan gerealiseerd worden in een SQL*Forms-applicatie die de gegevens opslaat in tabellen.
Het vastleggen van de gegevens heeft betrekking op:

- De benodigde tabellen, met de bijbehorende sleutelkolommen; Dit wordt in DDHULP vastgelegd;
- De relaties tussen deze de tabellen; welke tabel treedt als OUDERTABEL op en welke als KINDTABEL; Welke procedures gegenereerd moeten worden aan de OUDER-kant en welke aan de KIND-kant;
  Dit wordt in DDREL1 vastgelegd;
- Wat zijn de gerelateerde kolommen, met andere worden, het vastleggen van de sleutelkolommen en de relatiekolommen; Dit wordt in DDREL2 vastgelegd.

Voor de begrippen OUDERTABEL, KINDTABEL, sleutelkolommen en relatiekolommen zie [HELD] blz. 25.

De tabelstructuur van DDHULP, DDREL1, en DDREL2 worden hieronder weergegeven. De sleutelkolommen zijn onderstreept.

```
DDHULP : TABELNAAM                    char not null;

DDREL1 : OUDERTABEL                   char not null,
         KINDTABEL                    char not null,
         OUDER_INVOEGPROCEDURE        char not null,
         KIND_INVOEGPROCEDURE         char not null,
         OUDER_VERWIJDERPROCEDURE     char not null,
         KIND_VERWIJDERPROCEDURE      char not null;

DDREL2 : OUDERTABEL                   char not null,
         KINDTABEL                    char not null,
         OUDERKOLOM                   char not null,
         KINDKOLOM                    char not null.
```

Merk op dat de mutatieprocedure is direct af te leiden uit de gekozen invoegprocedure. Daarom wordt deze niet meer expliciet in kolom de DDREL1 gegeven.

In Bachman-diagram zien de relaties tussen DDHULP, DDREL1 en DDREL2 als volgt uit:

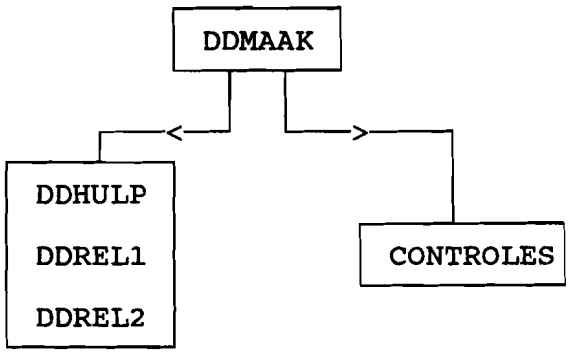
figuur 2: Bachman-diagram van DDHULP, DDREL1 en DDREL2.

Daarnaast dient DDMAAK ook controles uitvoeren.
Deze heeft betrekking tot de volgende:

a). Controleren of de tabellen bestaan;
b). Controleren of de tabelnamen in DDREL1 en DDREL2 in DDHULP staan;
c). Controleren of in elke relatie elke sleutelkolom aan de OUDER-kant aan bod komt, en of er geen alternatieve sleutel zijn;
d). Controleren of de gekozen procedures zijn wel aan elkaar gerelateerd in de zin dat ze allen overeenkomen met de specificatie van de relatie;
e). Controleren of binnen de relatie de relatiekolommen disjunct zijn;
f) Controleren of de kolomnamen bestaan;
g). Controleren of de tabelnamen en de kolomnamen toegestaan zijn;
h). Controleren of de tabelnamen in hoofdletters zijn.

Controles a),b) en d) worden bij elke poging om DDMAAK te verlaten
uitgevoerd.
Controles c) en h) worden door de programmatuur automatisch afgedwogen.
Controles e), f), en g) worden bij het invoeren van de tabel- en kolomnamen uitgevoerd.

Het structuurdiagram van de DDMAAK wordt hieronder weergegeven:



figuur 3:Het structuurdiagram van DDMAAK.

3.4 De Functionaliteit en Structuur van GENERATOR

Voordat de GENERATOR gestart wordt, moeten het te genereren applicatie-formulier in de SQL*Forms-database en de benodigde gegevens in de DDMAAK vastgelegd zijn.
De GENERATOR interpreteert de gegevens in de DDMAAK, en vervolgens gebruikt deze gegevens om de gewenste procedures in het gewenste applicatie-formulier te genereren.
Deze functie van "het interpreteren van gegevens" wordt met BESTURING aangeduidt. Deze BESTURING stuurt als het ware het generatieproces. Deze bepaalt welke gewenste procedures zijn en in welke tabel deze procedures moeten terecht komen.
Om deze te realiseren moeten de records in DDHULP doorlopen worden en voor elke relatie van deze tabellen.
Er onstaan dan als het ware repetities "voor elke tabel" en "voor elke relatie".

Vooraan begint BESTURING met initialisatie. Deze heeft betrekking op onder andere de commit-teller.

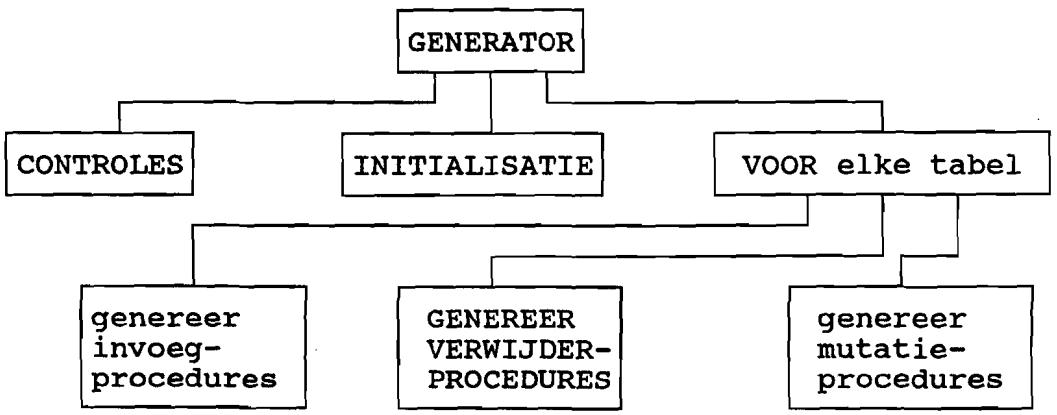
Daarnaast voert GENERATOR een aantal controles.
Deze controles hebben betrekking op het volgende:

- controleren of er wel een te genereren applicatie gekozen is;
- Controleren of de gebruiker van de GENERATOR de eigenaar is van de te genereren applicatie;
- Controleren of er al een procedure in de te genereren applicatie aanwezig is.

De eerste twee controles behoren tot de taken van BESTURING.
De laatste controle wordt opgenomen in verdere functie van de GENERATOR namelijk bij het "genereer van de gewenste procedures".
Deze controle wordt gerealiseerd met behulp van een booleanse vlag er_is_trigger.

Als alle controles goed zijn verlopen dan begint GENERATOR met het "genereren van de gewenste procedures".

Het structuurdiagram van de GENERATOR ziet dan als voglt uit:



figuur 4: Het structuurdiagram van GENERATOR.

De structuurtekst van de GENERATOR met meer specifiek m.b.t. "Genereer verwijderprocedures" :

```
GENERATOR
    CONTROLES;
    INITIALISATIE;
    VOOR "elke tabel"
    BEGIN
        "Genereer invoegprocedures";
         er_is_trigger := 'FALSE';
         VOOR "elke relatie"
             "Genereer verwijderprocedure";
        "Genereer mutatieprocedures";
    END.
```

```
Genereer verwijderprocedure;
   BEGIN
     CASE DDHULP.TABELNAAM IS
          DDREL1.OUDERTABEL : genereer Ouder_verwijderprocedure;
          DDREL1.KINDTABEL  : genereer Kind_verwijderprocedure;
     END CASE;
   END;
```

```
Genereer Ouder_Verwijderprocedure;
BEGIN
 CASE DDREL1.OUDER_VERWIJDERPROCEDURE IS
  TOEGESTAAN        :genereer Ouder_verwijderprocedure_toegestaan;
  BEPERKT           :genereer Ouder_verwijderprocedure_beperkt;
  VOLLEDIGE CASCADE:genereer Ouder_verwijderprocedure_vollcas;
  BEPERKT CASCADE   :genereer Ouder_verwijderprocedure_bepcas;
  NULLIFICEREN      :genereer Ouder_verwijderprocedure_nullifi-
                      ceren;
 END CASE;
END;
```

4. INTEGRITEITSPROCEDURES

4.1. Mogelijke en Wenselijke Procedures

Er zijn drie belangrijke acties mogelijk bij het onderhouden van een database: Invoegen, Verwijderen, en Muteren.
De integriteitsprocedures moeten dan zorgen dat tijdens deze acties, de referentiële integriteit van de database gehandhaafd blijft.
Deze integriteitsprocedures hebben verschillende eigenschappen, met betrekking tot de manier waarop ze de referentiële integriteit bewaken.
Deze acties hebben betrekking op de OUDERTABEL of op de KINDTABEL.

Een relatie waarbij elk record in de OUDERTABEL altijd tenminste één bijbehorend record in de KINDTABEL moet hebben, wordt met "KIND-verplicht" gespecificeerd en anders met "KIND-optioneel".
Een relatie waarbij elk record in KINDTABEL altijd (precies) één bijbehorend record in de OUDERTABEL moet hebben, wordt met "OUDER-verplicht" gespecificeerd en anders met "OUDER-optioneel".
Een relatie wordt gespecificeerd met "OUDER-beperkt-optioneel" wanneer de relatiekolommen in de KINDTABEL niet alle ingevuld hoeven te worden, waarbij elk record in de ingevulde relatiekolommen een bijbehorend record in OUDERTABEL moet hebben.

Hieronder volgt een opsomming van de mogelijke integriteits-procedures. Bij de verwijderprocedures wordt er tevens het verband tussen de relatie-specificatie en de te kiezen integriteitsprocedure aangegeven:

Invoegprocedures:

a). OUDERTABEL:
- * Toegestaan : Deze laat de invoeging altijd toe.
- * Beperkt : Deze laat de invoeging alleen toe als er nog minstens een bijbehorend KIND-record bestaat.
- * Neerwaartse-invoegcascade: Deze laat de invoeging alleen toe als er nog tenminste een bijbehorende KIND-record bestaat of tegelijkertijd wordt ingevoegd.

b). KINDTABEL:
- * Toegestaan : Deze laat de invoeging altijd toe.
- * Beperkt : Deze laat de invoeging alleen toe als het bijbehorende OUDER-record bestaat.
- * Opwaartse-invoegcascade: Deze laat de invoeging alleen toe als het bijbehorende OUDER-record bestaat of tegelijkertijd wordt ingevoegd.

* Beperkt-opwaartse-invoegcascade:
  Deze laat de invoeging alleen toe als het bijbehorende OUDER-record bestaat, of tegelijkertijd wordt ingevoegd, of als de referentiële velden niet allen gevuld zijn.
* Niet-leeg-beperkt:
  Deze laat de invoeging alleen toe als het bijbehorende OUDER-record bestaat, of als de referentiële velden niet allen gevuld zijn.

Verwijderprocedures:

a). OUDERTABEL:
* Toegestaan : Deze laat de verwijdering altijd toe.
* Beperkt : Deze laat de verwijdering alleen toe als er geen bijbebehorende KIND-records bestaan.
* Volledige-cascade:
  Deze procedure moet zorgen dat bij een verwijdering van een record in OUDERTABEL, ook alle bijbehorende KIND-records,als deze bestaan, worden verwijderd.
* Beperkt-cascade:
  Deze laat de verwijdering alleen toe als er ten hoogste één bijbehorend KIND-record bestaat en dat dat KIND-record dan ook verwijderd moet worden.
* Nullificeren: Deze procedure moet zorgen dat bij een verwijdering van een record in OUDERTABEL een aantal referentiële velden van de bijbehorende KIND-records leeg gemaakt worden.

b). KINDTABEL:
* Toegestaan : Deze laat de verwijdering altijd toe.
* Beperkt : Deze procedure moet zorgen dat de verwijdering alleen toegestaan indien na de verwijdering het bijbehorende OUDER-record toch nog over bijbehorend KIND-record beschikt in de KINDTABEL waar de verwijdering plaatsvindt.
* Opwaartse-cascade:
  Deze procedure moet zorgen dat indien bij de verwijdering van een KIND-record tot gevolg heeft dat het bijbehorende OUDER-record geen bijbehorend KIND-record meer heeft,dan wordt dat OUDER-record ook verwijderd.

Het verband dat bestaat tussen relatiespecificaties en de mogelijke verwijderprocedures wordt hieronder weergegeven.

RELATIESPECIFICATIE	PROCEDURES bij verwijdering in KINDTABEL
KIND-verplicht	Opwaartse cascade Beperkt
KIND-optioneel	Toegestaan

RELATIESPECIFICATIE	PROCEDURES bij verwijdering in OUDERTABEL
OUDER-verplicht	**Beperkt** Volledige cascade Beperkte cascade
OUDER-optioneel	Toegestaan
OUDER-beperkt-optioneel	Nullificeren

Mutatieprocedures:

De mogelijke mutatieprocedures zijn in principe te vormen van alle mogelijke combinatie van invoegprocedures en verwijderprocedures. In dit verslag wordt daarop niet verder ingegaan.

4.2. De verwijderprocedure OUDER-BEPERKT

De verwijderprocedure OUDER-BEPERKT moet dan zorgen dat de verwijdering bij OUDERTABEL alleen kan plaats vinden als het OUDER-record geen bijbehorende KIND-records meer heeft. Er zijn verschillende manieren wat betreft het tijdstip waarop de uitvoering van de procedure plaats vindt. De inhoud van de procedure hangt ook af van de manier waarop deze uitgevoerd wordt. Hieronder worden deze manieren beschreven met de termen die aansluiten bij de terminologie zoals die in de SQL*Forms gebruikt worden.

1). De procedure wordt uitgevoerd op het moment dat de gebruiker een record voor de verwijdering markeert. Dit wordt met term "DELREC" aangegeven.
2). De procedure wordt voor elk te verwijderen record uitgevoerd direct voordat de ermee corresponderende rij uit de database wordt verwijderd.
Dit wordt met "PRE-DELETE" aangegeven.
3). De procedure wordt uitgevoerd direct nadat de verwijdering in de database plaats vindt.
Dit wordt met "POST-DELETE" aangegeven.

Er zijn een aantal criteria gehanteerd, waaraan een keuze gemaakt kan worden:

a). De toepasbaarheid;
b). De mate van ingewikkeldheid;
c). Hoeveelheid uit te voeren statements;
d). De mate van het aansluiten bij de uitgangspunten;
e). De mate van het aansluiten bij de handmatige implementatie;
f). De mate van de overeenkomst tussen de inhoud van de database en het beeldscherm;
g). Betrouwbaarheid.

De overwegingen zijn de volgende:
ad a). Mogelijkheden 2) en 3) hebben bredere toepaasbaarheid; Mogelijkheid 1) is niet geschikt voor andere verwijder-procedures met name bij CASCADE en NULLIFICEREN;
ad b). 1) is iets meer ingewikkerder dan 2) en 3);
ad c). 1) heeft meer statements nodig dan 2) en 3);
ad d). Alle drie sluiten goed aan de bij de uitgangspunten (zie [HELD] §1.7);
ad e). Mogelijkheden 2) en 3) sluiten beter aan bij de handmatige implementatie;
ad f). Bij 1) is de overeenkomst tussen de inhoud van de database en het beeldscherm beter dan bij 2) en 3).
ad g). Het is de bedoeling dat de verwijdering niet door gaat indien er nog KIND-record behorend bij het OUDER-record. Hier heeft mogelijkheid 2) voordeel ten opzichte van 3); Mogelijkheid 2) wordt namelijk uitgevoerd terwijl de verwijdering in database nog niet plaats is gevonden;

Op grond van de criteria en de overwegingen wordt gekozen voor de mogelijkheid 2), de PRE-DELETE implementatie.

PRE-DELETE is (bij SQL*Forms) een commit-trigger. Deze wordt uitgevoerd wanneer de gebruiker een transactie naar de database wil committen.
Op het moment dat de gebruiker het record markeert (door delrec-functietoets indrukken) verdwijnt het record vervolgens uit het beeldscherm. Als dat record toch niet verwijderd mag worden, dan merkt de gebruiker dit pas bij het committen (commit-functietoets indrukken). Er komt dan een foutmelding.
Dit is echter het nadeel van de gekozen implementatie.
Maar dit nadeel is nog aannemelijk gezien van dat dit verschijnsel in SQL*Forms-applicatie veel voor kan komen.

Een combinatie van 1) en 3) is mogelijk, vooral om het nadeel van 2) en 3) over het overeenkomsten tussen het beeldscherm en de inhoud van de database weg te halen, maar daardoor neemt de ingewikkeldheid toe.
Deze combinatie blijft een goed alternatief.

De verwijderprocedure OUDER-BEPERKT bevat een controle of er bij het te verwijderen OUDER-record nog (een of meer) bijbehorende KIND-records bestaan.
Deze controle kan worden gedaan met behulp van SELECT-statement.

Ieder record in een tabel wordt (door ORACLE automatisch) uniek geïdentificeerd door zijn ROWID. Dat geldt ook voor het te verwijderen record.
Het gebruik van ROWID is hier, omwille van efficiency, noodzakelijk, want de relatie kan betrekking hebben op meerdere sleutelvelden bij de OUDERTABEL en meerdere relatievelden bij de KINDTABEL.

De inhoud van de relatievelden in KINDTABEL wordt vergeleken met de inhoud van de sleutelvelden in OUDERTABEL, wat betreft de ROWID van het te verwijderen record. De statement om vast te stellen of bij het actuele record in de OUDERTABEL een bijbehorend KIND-record bestaat, ziet er dan als volgt uit.

```
Select 'x'
from    KINDTABEL
where  (relatieveld1, relatieveld2, ...) in
       (select sleutelveld1, sleutelveld2, ...
        from   OUDERTABEL
        where  ROWID = :OUDERTABEL.ROWID )
```

Het dubbele punt voor OUDERTABEL.ROWID duidt aan dat het om het actuele record in de OUDERTABEL gaat.

Deze statement moet uitgevoerd worden als PRE-DELETE bij de OUDERTABEL.
Als deze "select"-statement succesvol verloopt dan betekent dat dat er nog bijbehorend KIND-record bestaat, dus mag het OUDER-record niet verwijderd worden. Er moet dan een foutmelding komen.

De specificatie ziet dan als volgt uit:

```
PRE-DELETE [OUDERTABEL]

Select 'x'
from    KINDTABEL
where  (relatieveld1, relatieveld2, ...) in
       (select sleutelveld1, sleutelveld2, ...
        from   OUDERTABEL
        where  ROWID = :OUDERTABEL.ROWID )


SUCCES ?  Ja -----> FOUT, er bestaat nog een bijbehorend
                           KIND-record;
          Nee ----> OK.
```

5. IMPLEMENTATIE

De implementatie vindt plaats in een aantal stappen (approximaties).

Zoals in § 3.4 vermeld wordt, dient de GENERATOR te controleren of er al een trigger aanwezig is in de te genereren applicatie. Wat deze controle betreft kan deze in nivo "genereer Ouder_verwijderprocedure_beperkt" gerealiseerd worden met behulp van IF-statement en de variabele er_is_trigger.

Bij de eerste approximatie wordt de specificatie van de integriteitsprocedure verwijderprocedure Ouder-Beperkt uitgebreid met een IF-statement:

```
"IF er geen trigger is THEN maak de volgende:"                @@@@

PRE_DELETE [OUDERTABEL]                                          @

Select 'x'                                                      ┐
From  KINDTABEL                                                 │
where (relatieveld1,relatieveld2, ...) in                       │ @@
      (select  sleutelveld1,sleutelveld2, ...                   │
       from OUDERTABEL                                          │
       where rowid = :OUDERTABEL.rowid)                         ┘

SUCCES ?  JA -----> FOUT, er bestaat nog bijbehorend            ┐
                          KIND-records;                         │
                          Afbreken, geef melding;               │ @@@
          Nee ----> OK.                                         ┘
```

@ : Heading van de trigger;
@@ : Body van de trigger ofwel de triggerstep;
@@@ : Attributen van de triggerstep; Hier wordt aangegeven hoe gereageerd moet worden op de uitkomst van de triggerstep.
@@@@ : Is een IF-statement.

In SQL*Forms-omgeving worden de procedures aangeduid met term "triggers", en de statements in de procedures worden aangeduid met term "triggersteps".

Er zijn een aantal hulpprocedures beschikbaar die de te genereren procedures met de daarin horende statements daadwerkelijk genereren. Deze hulpprocedures, met de daar bijbehorende parameters, zijn:

- Creer_trigger (tabelnaam,veldnaam,triggernaam);
  Met deze procedure wordt de heading van de trigger gegenereerd. De parameters tabelnaam en veldnaam geven aan: de tabel, en het veld van die tabel, waarbij de trigger gegenereerd moet worden.
  De parameter triggernaam geeft aan de naam van de trigger.

- Creer_triggerstep (afbreken,inverse,rollback,label,
succeslabel,foutlabel,foutmelding);
Met deze procedure worden de attributen van de triggerstep gegenereerd;
De parameters afbreken, inverse, en rollback functioneren als vlaggetjes. De parameters afbreken en inverse corresponderen respectievelijk met de triggerstep-attributen "abort trigger when step fails" en "reverse return code" in SQL*Forms-applicatie.
De parameter rollback geeft aan of de databasehandelingen (bijvoorbeeld invoeging, verwijdering of mutatie) geannuleerd moeten worden.
De parameters label, succeslabel, en foutlabel dienen ter realisatie van de sprongopdracht.
De parameter foutmelding spreekt van zelf, deze geeft aan de foutmelding-tekst.

- Creer_sqltextregel(text)
Deze hulpprocedure genereert de body van de trigger ofwel de triggerstep.

Zo kan als tweede approximatie de volgende gespecificieerd worden:

```
"IF er_is_trigger='FALSE' THEN"
Creer_trigger (OUDERTABEL, , 'PRE-DELETE');

Creer_sqltextregel (Select 'x'
                    From  KINDTABEL
                    Where (relatieveld1,relativeld2, ...) in
                        (select sleutelveld1,sleutelveld2, ...
                         from OUDERTABEL
                         where rowid = :OUDERTABEL.rowid)     );

Creer_triggerstep ('AFBREKEN','INVERSIE','ROLLBACK,' ',' ',' ',
                   'Bij dit OUDERTABEL-record bestaat nog
                   bijbehorend KINDTABEL-record.');
```

In SQL*Forms kan een IF-statement gerealiseerd worden met behulp van de macro-commando CASE ;

Een repetitie-statement kan met behulp van de macro-commando CASE en het voorzien van labels gerealiseerd worden;

Om alle gerelateerde kolommen op te nemen, moet de tabel DDREL2 doorlopen worden.
Deze kan worden gerealiseerd met behulp van een repetitie-statement en een aantal daarvoor beschikbare procedures, namelijk:

- query (tabelnaam);
  Om de eerste record uit de tabel op te halen;
- nextrec (tabelnaam);
  Om de volgende record uit de tabel op te halen.

De tabel waarvoor de procedure gegenereerd wordt is actueel in DDHULP. De onderhevige relatie is actueel in DDREL1 en DDREL2. Het gebruik van actuele records wordt aangegeven door dubbelpunt (':') vooraan de tabelnaam.

De string characters staan, zoals gebruikelijk, tussen de apostroffen.

De derde approximatie ziet dan als volgt uit:

```
Case er_is_trigger is
    'FALSE' : creer_trigger (:DDTAB.TABELNAAM,' ','PRE-DELETE');
              er_is_trigger := 'TRUE';
End case;


Creer_triggerstep('AFBREKEN', 'INVERSIE', 'ROLLBACK',' ',' ',' ',
                  'Bij dit' :DDREL1.OUDERTABEL '-record
                   bestaat nog bijbehorend' :DDREL1.KINDTABEL
                  '-record.');


Creer_sqltextregel ('Select 'x' from' :DDREL1.KINDTABEL);
query (DDREL2);
creer_sqltextregel ('where (' :DDREL2.KINDKOLOM);
ZOLANG nextrec(DDREL2);
       creer_sqltextregel (',' :DDREL2.KINDKOLOM);
ENDZOLANG;
creer_sqltextregel (') in');
query (DDREL2);
creer_sqltextregel ('       (select ' :DDREL2.OUDERKOLOM);
ZOLANG2 nextrec (DDREL2);
        creer_sqltextregel (',' :DDREL2.OUDERKOLOM);
ENDZOLANG2;
creer_sqltextregel ('        from ' :DDREL1.OUDERTABEL);
creer_sqltextregel ('        where ROWID = :' :DDREL1.OUDERTABEL
                                                 '.ROWID');
```

In de GENERATOR zijn er twee soorten globale variabelen gedefinieerd.De eerste soort is alleen toegankelijk via macro-commando's. Deze variabelen worden voorafgegaan door een het voorvoegsel "global".
De tweede wordt gevormd door speciaal daartoe gecreëerde velden in formulieren. Bij de GENERATOR worden deze variabelen voorafgegaan door het voorvoegsel "VARIABELEN".
Met betrekking tot deze variabelen zijn er twee soorten implementaties van de assignment statement te onderscheiden:

- Als het om het toekennen van waarde aan een "global"-variabele gaat,dan wordt dit gerealiseerd met behulp van "#copy-" commando;
- Als het om het toekennen van waarden aan een "VARIABELEN"-variabele gaat, dan wordt dit gerealiseerd met behulp van "select- ...-into" commando.

In SQL*Forms wordt elke procedure-aanroep uitgevoerd via de macro-commando "#exemacro exetrg <triggernaam>".

De procedures "query (tabelnaam)" en "nextrec (tabelnaam)" zijn te implementeren in respectievelijk:
"#exemacro goblk <tabelnaam>; exeqry;" en
"#exemacro goblk <tabelnaam>; nxtrec;".

Concatenatie wordt in SQL*Forms gekenmerkt door "||".

In SQL*Forms zijn de booleanse waarden aangegeven door 'T' voor true en 'F' voor false.

De vierde approximatie is dan de uiteindelijke implementatie "Seq # .." geeft aan de stappen van de triggerstep;

De vierde approximatie:

Generatieprocedure OUDER_verwijderprocedure_BEPERKT

seq #1:
```
#exemacro case global.er_is_trigger is
             when 'F' then exetrg foutconditie;
           end case;
                 success_label: er_is_verwijdertrigger
```

seq #2:
```
Select lower (:DDREL1.OUDERTABEL),null, 'PRE-DELETE'
into VARIABELEN.blok,VARIABELEN.veld,VARIABELEN.type
from dual
                (* abort trigger when step fails)
```

seq #3:
```
#exemacro exetrg creer_trigger;
                (* abort trigger when step fails)
```

seq #4:
```
#copy 'T' global.er_is_trigger;
                (* abort trigger when step fails)
```

```
seq #5:                 label: er_is_verwijdertrigger
      Select 'N','Y','Y','Y',null,null,null,
             'Bij dit '||:DDREL1.OUDERTABEL|| '-record bestaat
              nog bijbehorend ' ||:DDREL1.KINDTABEL|| '-record.'
      into    VARIABELEN.curs,VARIABELEN.mve,VARIABELEN.inv,
              VARIABELEN.roll,VARIABELEN.label,VARIABELEN.slab,
              VARIABELEN.flab,VARIABELEN.msg
      from    dual
                      (* abort trigger when step fails)

seq #6:
      #exemacro exetrg creer_triggerstep;
                      (* abort trigger when step fails)

seq #7:
      Select 'Select ''X''from' ||' '||:DDREL1.KINDTABEL
      into  VARIABELEN.sqttext
      from  dual
                      (* abort trigger when step fails)

seq #8:
      #exemacro exetrg creer_sqltextregel;
                goblk DDREL2; exeqry;
                      (* abort trigger when step fails)

seq #9:
      Select 'where (' ||:DDREL2.KINDKOLOM
      into VARIABELEN.sqttext
      from dual
                      (* abort trigger when step fails)

seq #10:                label: ZOLANG
      #exemacro goblk DDREL2; nxtrec;
                      (* abort trigger when step fails)

seq #11:
      #exemacro case DDREL2.OUDERTABEL is
                   when 'ZZZZZZZZZZ' then exetrg foutconditie;
                   when ''           then exetrg foutconditie;
                 end case;
                      failure label: ENDZOLANG

seq #12:
      Select  :VARIABELEN.sqttext ||',''||' '||:DDREL2.KINDKOLOM
      into     VARIABELEN.sqttext
      from     dual
                      (* abort trigger when step fails)
                      success label: ZOLANG

seq #13:                label: ENDZOLANG
      Select  :VARIABELEN.sqttext ||') in'
      into     VARIABELEN.sqttext
      from     dual
                      (* abort trigger when step fails)
```

```
seq #14:
      #exemacro exetrg creer_sqltextregel;
                goblk DDREL2; exeqry;
                        (* abort trigger when step fails)

seq #15:
      Select '      (select'||' '||:DDREL2.OUDERKOLOM
      into   VARIABELEN.sqttext
      from   dual
                        (* abort trigger when step fails)

seq #16:                label: ZOLANG2
      #exemacro goblk DDREL2; nxtrec;
                        (* abort trigger when step fails)

seq #17:
      #exemacro case DDREL2.OUDERTABEL is
                  when 'ZZZZZZZZZZ' then exetrg foutconditie;
                  when ''           then exetrg foutconditie;
                end case;

                        failure label: ENDZOLANG2

seq #18:
      Select  :VARIABELEN.sqttext ||','||' '||:DDREL2.OUDERKOLOM
      into     VARIABELEN.sqttext
      from     dual
                        success label: ZOLANG2

seq #19:                label: ENDZOLANG2
      #exemacro exetrg creer_sqltextregel;
                        (* abort trigger when step fails)

seq #20:
      Select '       from'||' '||:DDREL1.OUDERTABEL
      into   VARIABELEN.sqttext
      from   dual
                        (* abort trigger when step fails)

seq #21:
      #exemacro exetrg cree_sqltextregel;
                        (* abort trigger when step fails)

seq #22:
      Select  '       where ROWID = :'
              || :DDREL1.OUDERTABEL ||'.ROWID)'
      into    VARIABELEN.sqttext
      from    dual
                        (* abort trigger when step fails)

seq #23:
      #exemacro exetrg creer_sqltextregel;
                        (* abort trigger when step fails)
```

6. EVALUATIE

6.1 Het verloop van de stage

Het stageverloop kwam min of meer overeen met het geplande tijdschema.Maar bij sommige aktiviteiten is er meer tijd aan besteedt dan gepland. Dat heeft vooral te maken met technische problemen, die de stagiair niet zelf kan oplossen. Hieronder wordt gesommeerd hoe de stage is verlopen. De tegengekomen problemen zijn ook vermeld.

- Bestuderen van de opdracht en leren omgaan met de apparatuur.
  De apparatuur die wordt gebruikt tijdens de stage is IBM 4381 met VM/CMS 1.4.0 als besturingssysteem;
  Besteden tijd : 7 middagen.
  Problemen : Bij het inloggen.

- Bestuderen van ORACLE-SQL*Plus en oefenen met deze;
  Besteden tijd : 5 middagen.
  Problemen : Bij het inloggen.

- Bestuderen van ORACLE-SQL*Forms en oefenen met deze;
  Besteden tijd : 4 middagen.

- Bestuderen van het generatieproces;
  Besteden tijd : 4 middagen.

- Opstellen van de integriteitsprocedure "verwijderprocedure OUDER-BEPERKT";
  Besteden tijd : 2 middagen.

- Implementeren;
  Besteden tijd : 3 middagen.

- Testen;
  Besteden tijd : 3 middagen.
  Problemen : Het beperkte beschikbaarheid van geheugen.

- Verslag maken en afronding;
  Besteden tijd : 8 middagen.

6.2 De Begeleiding

Tijdens de stage is de stagiair begeleid door de heer du Bois en de heer Bruning. De heer du Bois begeleid stagiair in de dagelijkse gang van zaken, en de heer Bruning heeft meer een coördinerende taak verricht.
Door intensieve, wekelijkse, besprekingen met de begeleiders is de stagiair snel op het goede spoor gekomen met het uitvoeren van de opdracht.
Bovendien voor problemen kon de stagiaire bij diverse andere mensen terecht om de problemen te verhelpen.

BIBLIOGRAFIE

[BENN] Bennett, K., Cronin, D., ORACLE-SQL*Forms Designer's Tutorial: version 2.0, 1987.

[BROC] Brock de, E.O., De Grondslagen van Semantische Databases, Academic Service, 1989.

[CRON] Cronin, D.J., Mastering ORACLE, Hayden Books, 1988.

[HELD] Heldoorn, Theo, Het Genereren van SQL*Forms-Applicaties, TUE, 1991.

[PERR] Perry, T.J., en Lateer, J.G., Werken met ORACLE, Sybex, 1989.

[WOLF] Wolf, G., Module DB2 - Praktikumhandleiding, Hogeschool Eindhoven, 1990.

[ZUSS] Zussman, J.U., Sachs, J., Gomez, G., ORACLE-SQL*Forms Designer's Reference, Juli 1987.