Programmageneratie met Oracle*CASE

ir. D.J. Dral
Stichting Technische en Fysische Dienst voor de Landbouw (TFDL-DLO)
Postbus 356, 6700 AJ Wageningen
Telefoon 08370-76600, telefax 08370-11312
e-mail d.j.dral@tfdl.agro.nl

Inleiding

Oracle*CASE is een tool die ondersteuning geeft in de verschillende fasen van het traject van systeemontwikkeling. Oracle*CASE biedt de mogelijkheid het logisch model, de entiteiten, relaties en functies voor een informatiesysteem te definiëren. Daarnaast kunnen ook tabel- en moduledefinities worden vastgelegd, waarbij Oracle*CASE behulpzaam kan zijn bij de transformatie vanuit het logisch model. De laatste ontwikkeling van Oracle*CASE is de mogelijkheid om schermen en rapporten te genereren vanuit de moduledefinities.

Oracle*CASE bestaat uit verschillende onderdelen. CASE*Dictionary is een applicatie waarmee de repository (=database) van CASE, ook wel de Dictionary genoemd, kan worden onderhouden via invulschermen. Er kunnen ook rapporten over de inhoud van de database mee worden aangemaakt. CASE*Designer is een grafische interface naar de Dictionary. Hiermee kan de Dictionary worden gevuld of gewijzigd, tijdens het tekenen van een Entiteit-Relatie-diagram of een Functiedecompositie. Met CASE*Generator kunnen schermen, en sinds de laatste versie ook rapporten, worden gegenereerd op grond van moduledefinities in de Dictionary.

De CASE*Dictionary kan gegevens opslaan van het logisch model (entiteiten, relaties, functies) en van het fysieke of technische model (tabellen, modules). CASE*Generator baseert zich op gegevens van het fysieke model, vandaar dat het logisch model in dit artikel niet aan de orde komt.

In dit artikel zal met name de generatie van schermen door Oracle*CASE worden behandeld. Daarbij zullen de mogelijkheden en beperkingen van de schermgeneratie worden beschreven aan de hand van een voorbeeldapplicatie. Verder zal worden ingegaan op reverse engineering en re-engineering, waarnaar bij de TFDL-DLO onderzoek is gedaan.

Mogelijkheden van schermgeneratie

Om de mogelijkheden van schermgeneratie duidelijk te maken, zal gebruik worden gemaakt van een eenvoudige voorbeeldapplicatie. De voorbeeldapplicatie heeft als doel het beheren van een organisatie, die bestaat uit afdelingen waarbij werknemers werken. De gegevens betreffende deze beide objecten worden vastgelegd in twee tabellen, waarvan de structuur in figuur 1 is weergegeven.

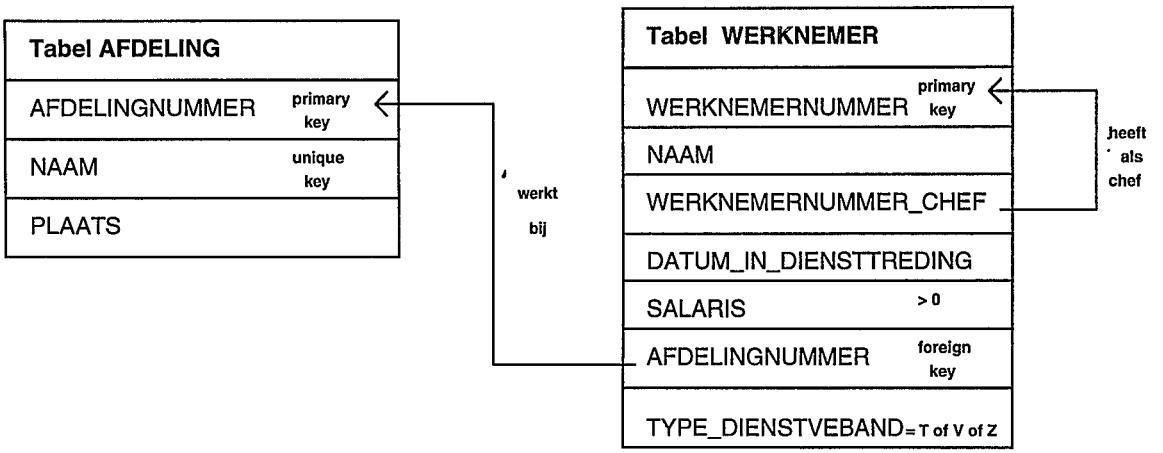
Gegevensdefinitie

De gegevens uit figuur 1 zijn opgeslagen in de CASE*Dictionary als tabel- en kolomdefinities. Behalve de namen van tabellen en kolommen worden datatypes en -lengten vastgelegd, die nodig zijn voor het aanmaken van de database. Samen met display prompts en help-regels kunnen die worden gebruikt bij de generatie van schermen.

De relaties en beperkingsregels worden vastgelegd als constraints. CASE kent verschillende typen constraints:

- De *primary key constraint* geeft de sleutel van de tabel aan. Deze kan uit een of meer kolommen van een tabel bestaan. Van de primary key wordt ook aangegeven of de kolommen mogen worden gewijzigd. Er kan een foutboodschap worden ingegeven, die wordt getoond aan de gebruiker, wanneer een rij wordt ingevoerd met een primary key die al bestaat in de database. Voor de tabel **AFDELING** zou dit bijvoorbeeld kunnen zijn: 'Afdeling met dit nummer bestaat al'.
- De *foreign key* geeft kolommen aan, waarvan de waarde moet voorkomen als primary key in een aangegeven tabel. Hierbij kan nog worden aangegeven wat moet gebeuren als de primary key, waarnaar wordt gerefereerd, wordt gewijzigd of als de rij met deze primary key wordt verwijderd. Het kan bijvoorbeeld zijn, dat verwijderen van de gerefereerde rij niet mag zolang er referenties naar die rij bestaan (restricted delete), of dat juist alle refererende rijen verwijderd moeten worden (cascading delete).
- Ten slotte kunnen bij de *check constraint* beperkingsregels t.a.v. de kolommen in een rij worden gedefinieerd, bijvoorbeeld 'DATUM_IN_DIENSTTREDING kleiner of gelijk aan systeemdatum' of 'BEGIN_DATUM kleiner of gelijk aan EINDDATUM'. Hier kunnen ook overige beperkingsregels, zoals die

Figuur 1 - Gegevensmodel voorbeeldapplicatie WAF



tussen meerdere rijen van een tabel, worden vastgelegd. Echter, controles hiervoor kunnen (nog) niet worden gegenereerd.

Voor kolommen kan worden vastgelegd wat geldige waarden zijn. Dit, in de vorm van een lijst van geldige waarden, een interval tussen minimum en maximum en combinaties van deze twee vormen. In de voorbeeldapplicatie is vastgelegd, dat TYPE_DIENSTVERBAND moet worden gekozen uit de waarden V, T en Z, die staan voor respectievelijk Vast, Tijdelijk en Z-formatie.

Moduledefinitie

De definitie van een module in CASE*Dictionary, i.e. een SQL*Forms-programma (SQL=Structured Query Language), is vrij summier. Behalve naam, omschrijving en type van de module hoeft alleen te worden vastgelegd welke tabellen en kolommen worden gebruikt door de module en welke mogelijkheden van manipulatie hiervan bestaan (opvragen, invoeren, wijzigen en/of verwijderen). Verder kan bij een veld een formaat, een schermtekst en help-tekst worden gedefinieerd.

Voor de module voor het beheren van afdelingen en werknemers definiëren wij een SQL*Forms-module Beheren Afdelingen en Werknemers (BAW). Bij deze module wordt het gebruik van alle velden van de tabellen **AFDELING** en **WERKNEMER** gedefinieerd met alle mogelijkheden van manipulatie. Bovendien wordt nog een gebruik van de tabel **WERKNEMER**, met de kolommen WERKNEMERNUMMER en NAAM, vastgelegd om informatie te bieden over de naam van de chef via AFDELING.WERKNEMERNUMMER_CHEF.

Generatie van de module

Bij generatie van de module wordt nog het verband tussen de gebruikte tabellen vastgelegd, waarna het scherm van figuur 2 wordt gegenereerd.

Behalve de header en footer van het scherm kunnen er twee blokken worden onderkend, het blok **AFDELING** en het multi-record-blok **WERKNEMER**. Deze blokken komen overeen met de twee tabellen die door de module worden gebruikt. De velden in beide blokken komen voort uit de definities van het gebruik van kolommen.

De twee blokken zijn verbonden via de foreign key van de tabel **WERKNEMER** naar de tabel **AFDELING**. Bij generatie is het mogelijk te kiezen tussen volledige en gedeeltelijke en geen synchronisatie. Een voorbeeld van 'volledige synchronisatie' is het direct tonen van werknemers wanneer een afdeling wordt opgevraagd. Gedeeltelijke synchronisatie daarnaast, is bijvoorbeeld het tonen van de werknemers van een afdeling wanneer naar het blok **WERKNEMER** wordt gesprongen.Bij geen synchronisatie moeten werknemers in het blok **WERKNEMER** apart opgevraagd worden.

De derde tabel, het tweede gebruik van **WERKNEMER**, komt tot uiting in de naam van de chef. Dit veld, dat niet verbonden is met een kolom uit de tabel werknemer, is niet betreedbaar, maar wordt gevuld wanneer een bestaand werknemernummer wordt ingevuld in het veld WERKNEMERNUMMER_CHEF.

Gegenereerde controles

De bij de tabellen gedefinieerde constraints en de beperkingen in mogelijke waarden worden gecontroleerd in het gegenereerde scherm:

- Er wordt gecontroleerd of een ingevoerd AFDELINGSNUMMER, WERKNEMERNUMMER en AFDELINGSNAAM uniek is. Deze kolommen zijn als primary key constraint en unique key constraint vastgelegd. De controle wordt naar keuze direct na invoer of bij vastleggen uitgevoerd.
- De inhoud van het veld WERKNEMERNUMMER_CHEF, een foreign key naar **WERKNEMER**, wordt gecontroleerd op bestaan als primary key van **WERKNEMER**. Op dit veld is een lijst van mogelijke werknemernummers beschikbaar.
- Omgekeerd zal het verwijderen van een afdeling worden verhinderd wanneer er nog werknemers zijn geregistreerd bij die afdeling. Deze controle is gebaseerd op de foreign key van

WERKNEMER naar AFDELING met restricted delete. Wanneer een cascading delete is gedefinieerd dan worden bij het verwijderen van een afdeling ook de werknemers, die bij deze afdeling zijn geregistreerd, uit de database verwijderd.

- De check constraint op DATUM_IN_DIENST leidt tot de controle, dat de opgegeven datum op of voor vandaag moet liggen.
- De mogelijk in te vullen waarden in TYPE_DIENSTVERBAND zijn beperkt tot de gedefinieerde waarden. De gebruiker kan een lijst van mogelijke waarden oproepen.

Wanneer de mogelijke manipulaties op een tabel zijn beperkt in de moduledefinitie in CASE, worden deze beperkingen ook gecodeerd in het gegenereerde scherm. Wanneer in de module BAW geen afdelingen mogen worden ingevoerd, gewijzigd of verwijderd, dan zal het alleen mogelijk zijn om afdelingen op te vragen. Alle andere acties zijn door gegenereerde triggers uitgesloten.

Overige mogelijkheden

De overige mogelijkheden bij schermgeneratie omvatten onder andere:

- *journalling tabellen.* Met CASE*Dictionary kan worden gedefinieerd of een tabel moet worden voorzien van een journaltabel, waarin alle wijzigingen worden bijgehouden. Bij generatie worden de schermen dan voorzien van code om deze journaltabellen te onderhouden.
- *afgeleide velden.* Het is mogelijk om afgeleide velden in de schermen te definiëren op grond van een formule waarin gerefereerd kan worden aan de kolommen van een record, bijvoorbeeld een veld JAARSALARIS=12*SALARIS, of BEDRAG=AANTAL*PRIJS. Ook wanneer een database-kolom is gedefinieerd als afgeleide kolom wordt door Generator code gegenereerd om deze kolom te onderhouden.
- *sommeringsvelden.* Op eenvoudige wijze kan een veld worden gedefinieerd voor bijvoorbeeld het totaal van salarissen binnen een afdeling. Dit totaal toont steeds direct het juiste bedrag, dus direct na de invoer van een nieuw salaris van een van de medewerkers wordt het totaal aangepast.

Sturing van schermgeneratie

Voor de sturing van de schermgeneratie kunnen zo'n 200 verschillende parameters, in CASE preferences genoemd, worden ingesteld. Dit kan op het niveau van de applicatie (het informatiesysteem), waarop indien gewenst per scherm wijzigingen kunnen worden ingegeven. De parameters worden opgeslagen in de Dictionary, zodat bij iedere keer generen de juiste preferences worden gebruikt. Belangrijke zaken waarop preferences onder andere invloed hebben zijn de layout van schermen, de functies van functietoetsen, het standaard datumformaat, het moment waarop controles worden uitgevoerd enz..

Indien gewenst kan gebruik worden gemaakt van een *template form*, een voorbeeld scherm waarop een aantal vaste elementen kunnen worden gedefinieerd en dat als basis dient voor te genereren schermen. Samen met de preferences kunnen hiermee bedrijfsstandaarden, voor wat betreft layout en formaten voor een aanzienlijk deel worden afgedwongen bij de generatie van schermen.

Beperkingen generatie

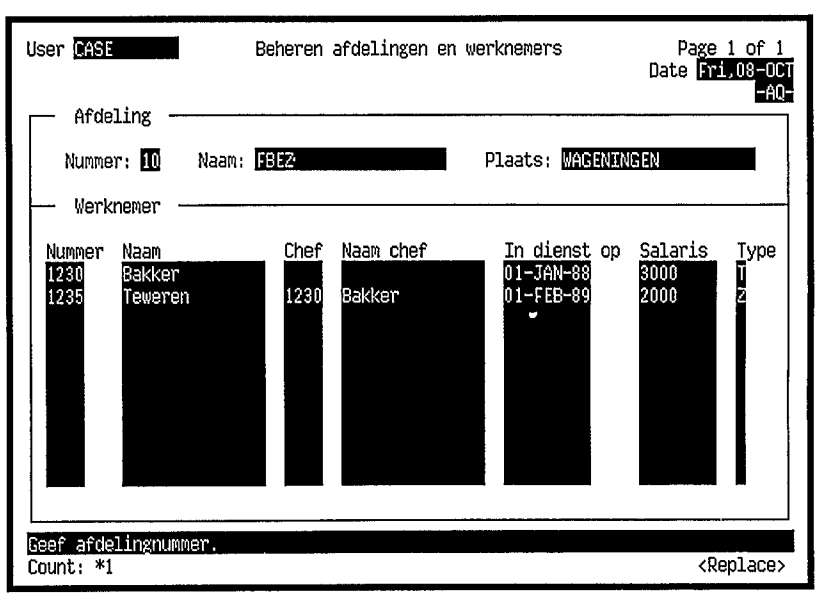
Nog niet alle functionaliteit kan worden gegenereerd. De belangrijkste beperkingen worden in deze paragraaf besproken.

Ten eerste, andere dan de genoemde constraints kunnen niet worden bewaakt. Men moet dan denken aan beperkingen die te maken hebben met meerdere rijen van dezelfde tabel of over meerdere tabellen heen. Een voorbeeld van de eerste is dat een werknemer niet meer mag verdienen dan zijn chef. Een voorbeeld van beperkingen over meerdere tabellen heen is dat een afdeling niet meer dan 50 werknemers mag hebben.

Ten tweede kan ook een complexe of afwijkende layout van schermen niet worden gegenereerd. Een blok kan bijvoorbeeld niet worden opgedeeld in twee delen door een ander blok. Dit zou wel gewenst kunnen zijn in een scherm voor het vastleggen van orders, waarbij gewenst kan zijn, dat het ordertotaal - onderdeel van het blok order -, onder het blok voor de orderregels wordt geplaatst.

Ten slotte kan proceslogica kan niet worden gedefinieerd en dus ook niet in schermen worden gegenereerd. Ook is het niet mogelijk om een controlblok, i.e. een blok zonder achterliggende tabel, te definiëren.

Figuur 2 - Door CASE*generator gegenereerd scherm



Hergeneratie

De in de vorige paragraaf genoemde beperkingen kunnen worden verholpen door via SQL*Forms aanpassingen aan de gegenereerde schermen uit te voeren. Wanneer hierbij geen door CASE*Generator gegenereerde PL/SQL-code wordt gewijzigd en de toegevoegde PL/SQL-code in aparte blokken wordt geplaatst, biedt CASE*Generator de mogelijkheid om het scherm te hergenereren. Bij hergeneratie wordt vooral de code ter controle van de beperkingsregels herzien. Eventueel nieuw gedefinieerd gebruik van tabellen en kolommen resulteert in blokken en velden op een nieuwe pagina, maar de definities van bestaande blokken en velden en de toegevoegde triggers en code worden niet aangetast.

Door deze genoemde eigenschappen is hergeneratie vooral geschikt voor het aanpassen van schermen bij een gewijzigd gegevensmodel. Bij gewijzigde moduledefinitie moeten vrijwel altijd nog aanpassingen op het resultaat worden uitgevoerd.

Evaluatie gebruik CASE*Generator

Een duidelijk voordeel van het gebruik van CASE*Generator is de tijdsbesparing bij het realiseren van schermen. Bovendien is de gegenereerde code betrouwbaarder, omdat deze op grond van vaste regels wordt gegenereerd (Let wel: de code kan niet betrouwbaarder zijn dan de definities). Dit betekent dat minder testinspanning benodigd is om een zelfde betrouwbaarheid te kunnen constateren.

Een voorwaarde voor succesvolle inzet van CASE*Generator is ervaring in het gebruik van CASE*Generator. Bij onervaren gebruikers kan veel tijd nodig zijn voor het vinden van de juiste definities en de juiste waarden voor de *preferences* om een bepaald resultaat te bereiken. Ook moet men door ervaring leren, welke functionaliteit het beste gegenereerd kan worden en welke functionaliteit beter handmatig kan worden toegevoegd.

Reverse engineering

Zoals uit het voorgaande blijkt biedt Oracle*CASE flinke ondersteuning bij de realisatie van schermen. Voorwaarde is wel, dat de definitie van database en schermen is vastgelegd in de Dictionary van CASE. Aan deze voorwaarde is voor de meeste bestaande systemen niet voldaan, zodat deze systemen niet zonder meer onderhouden kunnen worden met Oracle*CASE. Gedurende enige tijd biedt CASE al de mogelijkheid om de databasedefinitie in te lezen in de CASE*Dictionary. Na dit proces moet vooral op het gebied van constraints nog heel wat worden gedefinieerd, vooral als het gaat om een Oracle-database zonder constraints-definitie. In de nieuwste versie van Oracle*CASE bestaat nu ook de mogelijkheid om met een reverse engineering-hulpmiddel moduledefinities te vullen op basis van SQL*Forms-schermdefinities. Met dit hulpmiddel wordt een aanzienlijk deel van de moduledefinitie in CASE gevuld; een aantal zaken moet nog worden aangevuld. De controles op berperkingsregels kunnen echter niet worden vertaald in constraints en ook zaken, die niet in de Dictionary kunnen worden vastgelegd, zoals bijvoorbeeld proceslogica, gaan 'verloren' bij reverse engineering.

Voordelen van reverse engineering zijn tweeërlei. Aan de ene kant geeft een gevulde Dictionary een goede basis voor documentatie van een informatiesysteem. CASE biedt de mogelijkheid van *cross reference* en *impact analysis*. Er kan bijvoorbeeld een rapport worden gemaakt van alle modules, waarin een bepaalde tabel of kolom wordt gebruikt. Aan de andere kant geeft een systeem in CASE de mogelijkheid om dit systeem weer te genereren. Dit geldt in ieder geval voor de schermen, waardoor mogelijk een beter te onderhouden systeem ontstaat.

Een beperking van reverse engineering is dat het proces slechts definities van het technisch model vult. De definitie van het logisch model is een volledig handmatig proces.

Ervaringen met re-engineering

Bij de TFDL-DLO is een onderzoek uitgevoerd naar re-engineering mogelijkheden van Oracle*CASE. Met re-engineering wordt herbouw van een applicatie bedoeld door achtereenvolgens de definities van de applicatie te laden met reverse engineering om vervolgens de modules van de applicatie te genereren.

Als onderzoeksonderwerp werd de applicatie ROS gekozen, waarmee bij de TFDL-DLO leveringsopdrachten worden verwerkt. De definities van database en de schermen werden met reverse engineering in CASE geladen. Vooral de definitie van de database, uit Oracle versie 5, dus zonder constraint-definities, moest flink worden bijgewerkt. De aanvulling van de definitie van de schermen kostte niet zoveel tijd. Al met al scheelt de reverse engineering-optie heel wat tijd en geestdodend typewerk bij het inbrengen van de definitie van een bestaand systeem.

Na het definiëren van de applicatie in CASE werden de schermen van ROS gegenereerd. Een aantal schermen waren zonder veel moeite zo te genereren dat ze functioneel niet verschilden van de originele schermen. Anderen, met meer proceslogica, of bijzondere layout, moesten meer worden aangepast. Voor sommige schermen moesten redundante kolommen worden opgenomen in de tabellen of moest een 'view' worden gedefinieerd.

Uiteindelijk was de re-engineering van ROS een flinke operatie, die duidelijk meer tijd heeft gekost dan werd verwacht. Dit is gedeeltelijk te wijten aan het doel om de applicatie uiterlijk gelijk te laten zijn, gedeeltelijk aan de tabelstructuur van ROS, die niet zonder meer geschikt was voor het genereren van de gewenste schermen.

Uit het onderzoek is te leren, dat re-engineering met CASE een omvangrijke operatie is. Wellicht is het te overwegen om re-engineering uit te voeren in combinatie met groot onderhoud, omdat gebruikers voor hun gevoel niets winnen bij alleen re-

engineering. Ook is het remmend voor een re-engineering-proces, wanneer precies dezelfde schermen als in de bestaande applicatie moeten worden opgeleverd, dit kan nog heel wat aanpassingen na generatie vergen.

Een belangrijke reden om toch re-engineering uit te voeren is, dat het systeem eenvoudiger te onderhouden zal zijn. Bovendien zal de schermgenerator van CASE in de loop der jaren steeds meer functionaliteit krijgen, waardoor schermen steeds vollediger kunnen worden gegenereerd. Wanneer men hiervan wil profiteren zal toch op enig moment re-engineering moeten worden uitgevoerd om de applicatie voor de eerste keer te kunnen genereren.

Conclusie

Door de inzet van CASE*Generator kan de benodigde tijd voor realisatie van een informatiesysteem worden bekort. De testinspanning kan navenant worden verminderd, omdat de gegenereerde code betrouwbaarder is dan handgeschreven code.

Reverse engineering biedt de mogelijkheid om relatief makkelijk het fysieke model van een bestaande applicatie in CASE te definiëren. Re-engineering is een omvangrijke operatie, die beter in combinatie met groot onderhoud kan worden uitgevoerd dan als doel op zich.