Themaopdracht V2TOSAD

Thema: Software Architecture & Design

Opleiding: ICT-SIE

Versie: 27-10-2019

Inhoud

Dit project is tot stand gekomen in samenwerking met medewerkers van Transfer Solutions, te Leerdam.
Zo zijn bijvoorbeeld de code-voorbeelden aangeleverd door de heer Paul Koppens van Transfer Solutions.

© HU/Leo Pruijt

1. Introductie van het project

1.1 Het doel van de opdrachtgever

Oracle biedt een Business Rule generator aan genaamd "CDM-RuleFrame", waarmee database procedures en triggers gegenereerd kunnen worden in de vorm van (PL)SQL. De functionaliteit van deze generator volstaat goed en het tool is veelvuldig door software ontwikkelaars van verschillende bedrijven toegepast. Toch heeft een van deze bedrijven, Transfer Solutions uit Leerdam, gevraagd een opvolger voor het "CDM-RuleFrame" te ontwikkelen. En wel om de volgende redenen:

- Het CDM-RuleFrame werkt alleen in combinatie met Oracle Designer en Headstart. Designer is een modelleertool met codegeneratoren waarvan de eerste versies eind jaren tachtig op de markt kwamen. Oracle heeft inmiddels aangekondigd dat het Designer niet verder zal ontwikkelen, waardoor het niet aantrekkelijk meer is om nieuwe systemen met gebruik van Designer te ontwerpen.
- Voor het gebruik van Designer en Headstart moeten licentiekosten betaald worden.

Paul Koppens van Transfer Solutions licht de opdracht toe op:
http://www.weblectures.hu.nl/p2gplayer/Player.aspx?id=dkNF5K

1.2 De opdracht

Ontwikkel een open-source Business Rule generator die vergelijkbare functionaliteit levert als het CDM-RuleFrame.
De requirements en de specificaties van de te ondersteunen business rules staan verder op in dit document. Naast functionele eisen moet het tool ook aan een aantal niet-functionele eisen voldoen, die een brede toepasbaarheid (ook voor andere databases) en een lange levensduur mogelijk maken.

Zorg er voor dat:

- Alle requirements meegenomen worden bij het opstellen van de software architectuur en het ontwerp van de business rule generator.
- Het systeem gerealiseerd wordt conform de software architectuur en het ontwerp. Doe dat zoveel mogelijk model-gedreven en gebruik daarbij onder andere een Oracle database, Java en naar keuze ook APEX.
- De producten opgesteld worden conform de werkwijze, notatiewijze, tooling en kwaliteitscriteria, zoals behandeld in de andere cursussen van dit thema.
- Er methodisch en iteratief gewerkt wordt conform de systeemontwikkelmethode OpenUP.
- De user interface van het tool in de Engelse taal wordt vormgegeven en presentaties in het Engels worden verzorgd.

1.3 De leerdoelen

Deze themaopdracht biedt je de kans om de rol van software architect in een project te leren kennen en je vaardigheden m.b.t. design en programmeren op een hoger niveau te brengen.

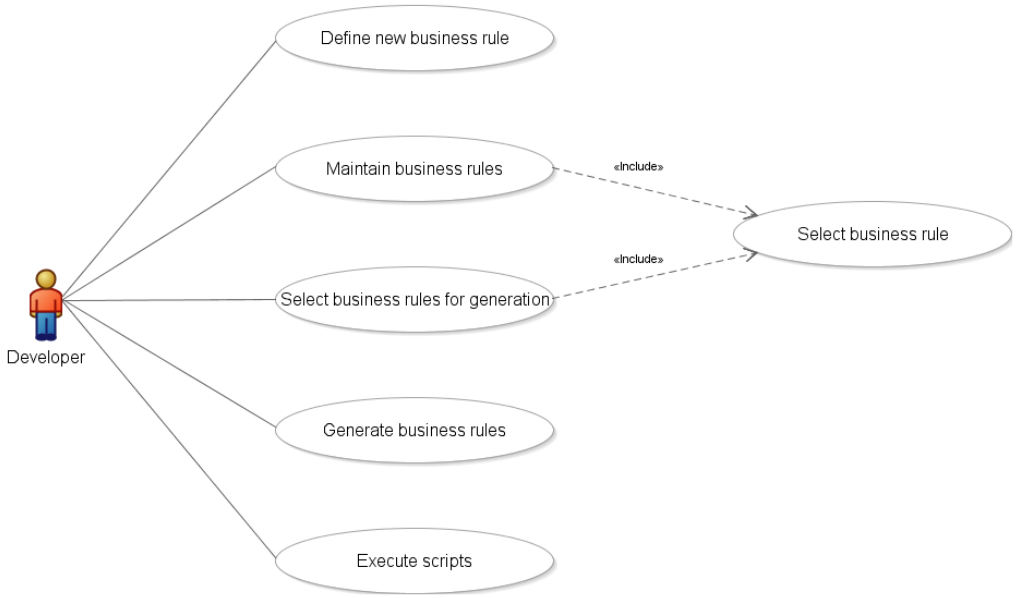
Het doel is dat je na afloop het volgende kan:

- Op basis van gegeven requirements en specificaties de key concepts identificeren, een informatiemodel opstellen en constraints op deze data specificeren.
- Op basis van gegeven requirements en specificaties belangrijke ontwerpbeslissingen nemen en verantwoorden en deze vastleggen in een Architecture notebook.
- De communicatie (functioneel en technisch) tussen de subsystemen ontwerpen.
- Op basis van gegeven requirements en specificaties functionele en technische ontwerpen opstellen, waarbij rekening wordt gehouden met de gemaakte architectuurkeuzen en keuzen voor de toepassing van design patterns.
- De juistheid van belangrijke ontwerpkeuzen aantonen aan de hand van een prototype.
- De software, bestaande uit meerdere subsystemen, realiseren (conform het ontwerp) en testen.
- De projectuitvoering organiseren, het werk verdelen, en effectief samenwerken, zodat het project tot een succesvol einde wordt gebracht.
- Het project en de persoonlijke rol binnen het team evalueren.
- In het Engels communiceren, zowel mondeling als schriftelijk.

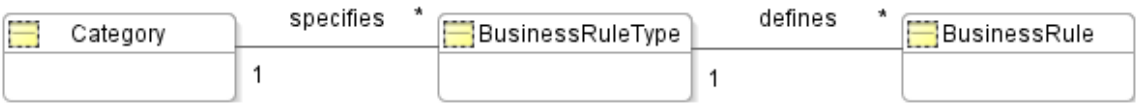
1.4 Initiële Software Architectuur

Zoals vastgesteld tijdens de Inception fase.

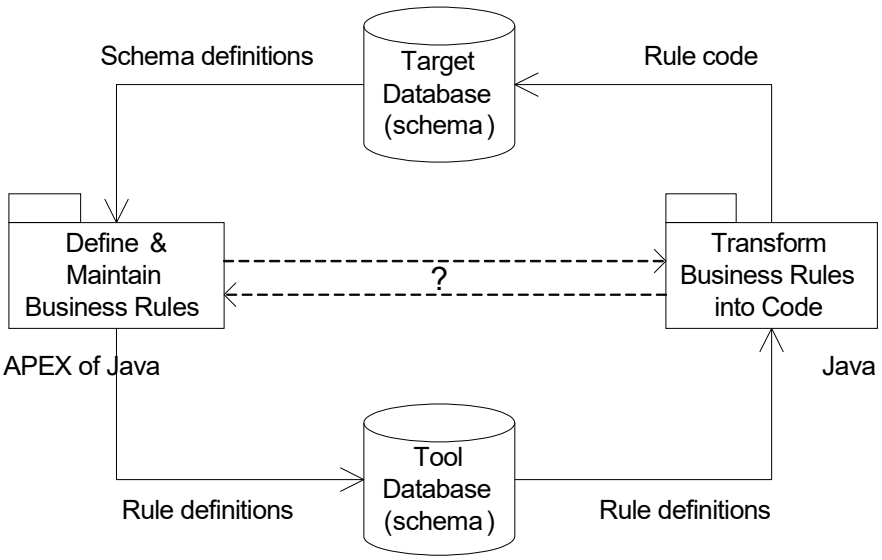
Use case model
Architectural significant use cases: 'Define new business rule', 'Generate business rule'



Domain model: Key abstractions



Software partitioning: Two subsystems (Define … and Transform …) in context



NB1 Subsystem Transform … moet met Java ontwikkeld worden. Voor subsystem Define … mag je kiezen voor APEX of voor een Java-backend in combinatie met een vrij te kiezen frontend technologie.

NB2 De "data flow" pijlen geven aan welke informatie in concept in welke richting gaat.
Voor de stippellijnen-pijlen moeten nog keuzen gemaakt worden.

1.5 Planning en producten

Tijdens de Inception, die voorafgaand aan dit project heeft plaatsgevonden, zijn de doelen en requirements vastgesteld, alsmede de initiële software architectuur, die de oplossingsrichting aangeeft.
De Elaboration en Construction fasen zijn, met de op te leveren producten, als volgt ingepland:

Week	Activiteiten	Producten
1	• Uitleg opdracht en vorming van teams • Presentatie: Dhr. Paul Koppens, Transfer Solutions • Start Elaboration Iteration 1 • Organiseer team en plan fase • Domein model opstellen	 Iteration plan Aanzet domeinmodel
2-3	• Producten opstellen Elaboration Iteration 1 • Overleg met begeleider a.d.h.v. iteration plan	

3-4	• Presentatie producten Elaboration Iteration 1
	  • Week 3: Define & Maintain Business Rules
	  • Week 4: Transform Business Rules into Code
	• Start Elaboration Iteration 2
	  • Plan fase
		Presentatie:
		Initieel Architecture notebook
		Functioneel werkingsmechanisme
		
		Iteration plan

5	• Producten opstellen Elaboration Iteration 2	
6	• Producten opstellen Elaboration Iteration 2	
P1	• Presenteren producten Elaboration Iteration 2	Architecture notebook Technisch ontwerp rapport Prototype
P1-P2	Construction fase Iteration 1 • Ontwerpen en realiseren van de applicatie • Systeemtest ontwerpen en uitvoeren • Demonstreren resultaat	Iteration plan Werkend systeem 1e iteratie Construction Report
P2-P3	Construction fase Iteration 2 • Ontwerpen en realiseren van de applicatie • Systeemtest ontwerpen en uitvoeren • Evaluatiegesprek met begeleider • Eindpresentatie voorbereiden en houden	Iteration plan Werkend systeem 2e iteratie Construction Report

1.6 Beoordeling

Er zijn meerdere beoordelingen die samen een eindcijfer vormen dat in Osiris ingevoerd wordt.
1. Elaboration 2: producten + presentatie: 50%
2. Construction 2: Systeem + Construction report + Presentatie: 50%

Tijdens de beoordeling van deze opdracht wordt onder andere gekeken of de Software Architectuur, het Design en de Implementatie: a) Een goed antwoord geven op *alle* gestelde eisen, dus ook op de niet-functionele eisen en de aanvullende architectuur eisen; b) Onderling consistent zijn.

Herkansing
Als je de themaopdracht net niet haalt, dan kun je in de eerste weken van blok C repareren. Je moet hiervoor afspraken maken met je vakdocenten. Deze reparatie telt als je tweede kans. haal je die niet, dan moet je het jaar erna de opdracht opnieuw doen. Als je niet mag of wil repareren, is er de mogelijkheid om de volledige opdracht te herkansen in de zomervakantie, Periode E. Je moet dan een compleet nieuwe, andere, themaopdracht maken ter herkansing. In plaats daarvan kan je natuurlijk ook het volgende studiejaar herkansen met de reguliere themaopdracht.

Plagiaat
Het zal je niet verbazen dat je deze opdracht zelfstandig moet uitvoeren, zonder gebruik te maken van (gedeeltelijke) uitwerkingen van andere teams of producten uit vorige jaren. Dit geldt ook herkansers.
Bij constatering van plagiaat wordt e.e.a. bij de Examencommissie gemeld en dat heeft flinke consequenties.

1.7 Procedures en materialen

- Ieder teamlid moet aanwijsbare producten ontwerpen, ontwikkelen en presenteren!
- Je mag aan het project werken wanneer je wil. Tijdens de ingeroosterde uren kan je in ieder geval in het practicumlokaal terecht. Het project 'weegt' 5 ECTS = 140 studiebelastinguren, dus je zal ook buiten de ingeroosterde tijdstippen aan de slag moeten.
- Belangrijke informatie staat op de cursus-site.

1.8 Samenwerken & Projectmanagement

Een goede samenwerking bepaalt het succes van het (ieder) project en het plezier dat je erin hebt. Een passende werkverdeling, duidelijke afspraken en een goede inzet en commitment van ieder projectlid zijn daarbij erg belangrijk.
Dit project geeft je keuzemogelijkheden welke competenties je wilt vergroten. Want er zal werk verdeeld moeten worden, omdat er te veel, te verschillende taken zijn om alles samen te doen. Verdeel daarom de volgende rollen (één persoon kan meerdere rollen vervullen):

- Project manager/Teamleider
  - Is verantwoordelijk voor de taakverdeling, de planning en uitvoering van het project.
  - Coördineert de oplevering van de producten, zodat ze op tijd en volledig zijn.
- Analyst/Analist
  - Stelt de analysemodellen en specificaties op.
- Architect
  - Stelt de architectuurmodellen en specificaties op.
  - Zorgt ervoor dat grote projectrisico' s vermeden worden. Zo nodig door het vroegtijdig ontwikkelen van prototypes.
  - Is verantwoordelijk voor het architecture notebook.
- Developer
  - Stelt het technisch ontwerp en de technische detailspecificaties op.
  - Realiseert de gevraagde functionaliteit.
  - Ontwerpt de developer tests (white box tests) en voert die uit.
- Tester
  - Ontwerpt de systeemtest (black box tests) en voert die uit.
- Team Database administrator en Repository owner
  - Beheert het database schema, de tabellen en (test)data van het team.
    En de toegang/autorisaties daartoe.
  - Beheert de ontwerpmodellen en specificaties.

Rik Boss vervult de DBA-rol m.b.t. de Oracle database. Neem contact met hem op als je vragen of problemen hebt.

2. Opdracht Elaboration Iteration 1

1. Verkrijg overzicht over het project en de eerste fase.
   - Lees de opdracht geheel door en verkrijg een overzicht over de fasen en producten.
   - Bestudeer de Requirements en de bijbehorende daaropvolgende hoofdstukken.
   - Bestudeer de initiële software architectuur, die in het vorige hoofdstuk is meegegeven.

2. Plan de fase.
   - Verdeel de rollen.
   - Stel een Iteration plan voor deze fase op (zie de template op de cursus-site).
     Bepaal de op te leveren producten en deel die zo mogelijk op (product breakdown structure).
     Creëer activiteiten om de deelproducten te ontwikkelen en koppel die aan teamleden. Creëer ook activiteiten om de deelproducten te integreren en de kwaliteit onderling te controleren.
     Bespreek het Iteration plan in de tweede of derde week met een begeleider.

3. Ontwerp een functioneel werkingsmechanisme voor de business rule generator.
   Analyseer de requirements m.b.t. de business rule generator (zie verder op in dit document) diepgaand. Ontwerp vervolgens een functioneel werkingsmechanisme voor de architectural significant use cases: 'Define new business rule' en 'Generate business rules'.
   Lever de onderstaande ontwerpproducten op en zorg voor consistentie tussen deze producten.
   3.1. Domein model
   Stel een gedetailleerd domeinmodel op (niveau functioneel ontwerp; zie de OOAD-lessen) in de vorm van een UML klassendiagram.
   Op basis van dit model moeten alle relevante eigenschappen vastgelegd kunnen worden van de business rules die in het systeem ingevoerd moeten kunnen worden. Dus ontwerp:
      - De relevante domeinklassen (gewoon, super, sub)
      - De attributen per klasse met hun type.
      - De associaties (naam/rollen, multipliciteit).

   3.2. Beperkingsregels m.b.t. het domeinmodel
   Beschrijf per type business rule een specificatie van de beperkingsregels die de waarden beperken die de ontwikkelaar in het tool mag invoeren bij het definiëren van een nieuwe busines rule (of het wijzigen van een bestaande rule). Beschrijf de rules die niet al in het domeinmodel vastliggen.
   3.3. Sequence diagrammen
   Ontwerp een functioneel werkingsmechanisme voor de ‘architectural significant use cases’. Stel sequence diagrammen op die het werkingsmechanisme (niveau functioneel ontwerp) laten zien. Welke klassen en objecten moeten wat doen om de use cases af te handelen? Doe dat voor de beide significante use cases.

4. Stel initiële versies van de onderstaande producten uit het Architecture Notebook op.
   4.1. Software partitioning model
   Ga uit van het software partitioning model in het vorige hoofdstuk en werk het verder uit. Ontwerp hoe de twee deelsystemen onderling zullen gaan communiceren.
      - Neem het diagram over en pas het aan (vervang o.a. het vraagteken en de stippelpijlen).
      - Specificeer in tekst:
         - Welke services bieden de deelsystemen aan? Welke data wordt uitgewisseld?
         - Met welke technologie wordt ieder deelsysteem geïmplementeerd? Waarom?
         - Hoe gaan de deelsystemen technisch communiceren? Met welke technologie?

   4.2. Tier-model
   Stel een grafisch model op en gebruik daarbij de termen van de Java Enterprise Edition tiers (zie SARCH les SA 2). Maak duidelijk welke tiers onderscheiden worden, hoe de functionaliteiten technisch over de tiers verdeeld worden en hoe de communicatie verloopt.

5. Presenteer de bovenstaande producten (in het Engels) en licht die toe.
   Alle teams presenteren (een deel van) hun producten. De andere teams en de docenten kunnen daarop verbeteringen voorstellen. Het is de bedoeling dat kennis en inzicht gedeeld worden, zodat alle teams tijdens de Elaboration een goede architectuur kunnen ontwerpen.
   - Zorg voor goed leesbare producten, dus houd daar in de layout rekening mee.
   - Zorg ervoor dat je de producten met een beamer kan presenteren.

3. Opdracht Elaboration Iteration 2

NB Om de juiste werking van de business rule generator te waarborgen, maar ook om de onderhoudbaarheid te vergroten, gelden de volgende **aanvullende architectuur eisen**:

Deelsysteem: Define and Maintain Business Rules

1) Alle altijd geldende beperkingsregels, met betrekking tot de data in de eigen database van de generator, moeten onder alle omstandigheden gecontroleerd worden. Ook als andere deelsystemen of applicaties data zouden toevoegen, muteren of verwijderen.
2) Er moeten drie logische componenten worden onderscheiden.
   a. Bij APEX-implementatie: die componenten moeten als PL-SQL packages in de database worden geïmplementeerd. Plaats daar bijbehorende procedures, functions, etc. in.
   b. Bij Java-implementatie: Onderscheid naast deze componenten (minstens) twee lagen.

Deelsysteem: Transform Business Rules into Code

1) Onderscheid twee componenten en (minstens) twee lagen. Specificeer de verantwoordelijkheid van deze software modulen en specificeer de architectuurregels die de modulen beperken.
2) Voor een stuk code van de generator dat specifiek is voor één type business rule moet duidelijk zijn voor welk type business rule deze code geldt.

1. Plan de fase en stel een Iteration plan op.
   Bestudeer OpenUP en stel een plan op gericht op: 1) verfijning van het werkingsmechanisme van de business rule generator; 2) het Architecture Notebook; en 3) een werkend prototype (proof of concept).

2. Ontwikkel de benodigde ontwerpproducten.
   2.1. Architecture Notebook (één, voor het gehele systeem)
   Neem de volgende onderdelen op:
   1. Architectural significant requirements
   2. Decisions & Justification
      Verantwoord in ieder geval de keuzen die zijn gemaakt om te voldoen aan de non-functional requirement NFR3, NFR4 en NFR5, en aan de bovenstaande **aanvullende architectuur eisen**.
   3. Domain model
      3.1. Domein klassendiagram (verbeterd; zie Elaboration 1).
      3.2. Beperkingsregels (verbeterd; zie Elaboration 1).
   4. Software partitioning model
      4.1. Top-level model
      Ga uit van het software partitioning model in het vorige hoofdstuk en werk het verder uit. Ontwerp hoe de twee deelsystemen onderling zullen gaan communiceren.
      - Neem het diagram over en pas het aan (vervang o.a. het vraagteken en de stippelpijlen).
      - Specificeer in tekst:
        - Met welke technologie wordt ieder deelsysteem geïmplementeerd? Waarom?
        - Hoe gaan de deelsystemen technisch communiceren?
          Welke services bieden de deelsystemen aan? Met welke technologie?

      4.2. Voor ieder deelsysteem: Aparte modellen van de modulaire architectuur.
      Ontwerp deze architectuur voor beide deelsystemen (Define … en Transform …).
      4.2.1. Leg deze architecturen vast in de vorm van UML package en/of component diagrammen, waarin te zien is welke modulen er zijn en welke afhankelijkheden tussen de modulen zijn toegestaan.
      Vul het diagram aan met een specificatie van: a) de klassen per component; b) de verantwoordelijkheden per laag;
      c) hoe iedere component of laag fysiek geïmplementeerd is.
      4.2.2. **Stel je ontwerp zo op dat er ook wordt voldaan aan de non-functional requirement NFR3, NFR4 en NFR5, en aan de bovenstaande aanvullende architectuur eisen.**

      4.3. Voor ieder deelsysteem in Java: Resultaten van de architecure conformance check.
      Neem voor ieder Java deelsysteem een apart "*Intended* Architecture diagram" op, gemaakt met HUSACCT, dat de ontworpen modulen (lagen, componenten, …) laat

zien en de dependencies tussen deze modulen nadat de packages en klassen van het prototype assigned zijn aan deze modulen.
Voer ook een architecure conformance check uit en toon de resultaten (tabel, diagram).

2.2. Technisch Ontwerp
Stel het technisch ontwerp op, rekening houdend met de software architectuur, en leg dit vast in een rapport. Dit hoeft alleen voor de twee architectural significant use cases:
1. Define new business rule
   1.1. Ontwerp de database structuur.
   1.2. APEX: Geef per component/package voorbeelden van concrete functionaliteiten.
   1.3. Java: Beschrijf, of geef in een klassendiagram aan, waar welk design pattern is toegepast.

2. Generate business rules
   2.1. Stel een class diagram op met de te implementeren klassen, directionaliteit, etc.
   2.2. Stel een (of meerdere) sequence diagram(men) op en laat zien hoe de te implementeren klassen de use case afhandelen. Houd rekening met componentgrenzen.
   2.3. Beschrijf, of geef in een klassendiagram aan, waar welk design pattern is toegepast.

3. Ontwikkel werkende prototypes voor de 'architectural significant use cases'.
Bewijs daarmee dat de ontworpen architectuur en ontwerpmechanismen kloppen.
Voor beide prototypes geldt dat je nog niet aan alle functionele en niet-functionele eisen hoeft te voldoen; dat volgt tijdens de Construction. Biedt in ieder geval wel ondersteuning voor de Attribute Range rule (d.m.v. database trigger) en liefst nog een tweede type.
Zorg ervoor dat de ontworpen software architectuur (vooral het software partitioning model) gevolgd wordt en herkenbaar is binnen de software!
   3.1. Define new business rule
      3.1.1. Bewijs dat je een business rule kan vastleggen.
      3.1.2. Implementeer de software architectuur.
         - APEX: Creëer database packages om de logische componenten te implementeren, voeg inhoud toe en gebruik die inhoud.
           Nog niet alle beperkingsregels/controles hoeven te worden geïmplementeerd, maar wel enkele per package in het software partitioning model.
         - Java:
           Creëer packages die de modulaire architectuur duidelijk reflecteren en voeg de bijbehorende subpackages en/of klassen toe.
   3.2. Generate business rules
      3.2.1. Bewijs op een zo eenvoudig mogelijke wijze, dat met het door jou ontworpen mechanisme een business rule definitie ook echt kan worden omgezet in code. Richt je op het meest riskante gedeelte.
      Er hoeven nog geen data uit de database te worden opgehaald.
      3.2.2. Implementeer de software architectuur.
      Creëer packages die de modulaire architectuur duidelijk reflecteren en voeg de bijbehorende subpackages en/of klassen toe.

4. Presenteer de producten
   - Lever het Architecture Notebook en het Technisch Ontwerp rapport op tijd op, conform de planning in Canvas. Deze producten worden tijdens de sessie besproken en beoordeeld, dus zorg voor leesbare diagrammen, een goede ordening, een inhoudsopgave en paginanummers op de bladzijden, et cetera.
   - Presenteer het prototype in het Engels.
   - Laat tijdens de demonstratie van het prototype ook zien hoe de software architectuur (vooral het software partitioning model) geïmplementeerd is en welke patterns zijn toegepast.

4. Opdracht Construction (Iteration 1 en 2)

De Constructiefase is opgedeeld in twee iteraties van beide ongeveer een week. Aan het eind van iedere iteratie presenteer je de producten en worden deze producten ook beoordeeld. In de eerste iteratie hoeven alleen de relatief eenvoudige business rules ondersteund te gaan worden met invoer- en beheerschermen, validaties en de generatie van business rules. In de tweede iteratie volgt dan de rest. Zie voor een beschrijving welke typen business rules in welke Iteratie aan bod moeten komen de paragraaf met de specificaties van de business rule typen.
Het stappenplan voor beide iteraties is grotendeels gelijk. Let wel op de verschillen in de producten die voor de beoordeling moeten worden aangeleverd of gepresenteerd.

1. Plan de fase/iteratie.
   - Stel een Iteration plan voor deze fase op.

2. Pas de architectuur en ontwerpproducten uit de Elaboration aan op basis van feedback.

3. Breid het functioneel ontwerp uit met de niet-architectureel significante use cases.
   - Zorg er in ieder geval voor dat alle attributen, nodig voor deze use cases, in het domeinmodel en het databaseontwerp zijn opgenomen.
   - Zorg ervoor dat aan de beslissingen en de richtlijnen van de software architectuur gevolgd worden.

4. Ontwerp de tests.
   Specificeer testgevallen op basis van het definitieve Elaboration Report.
   Stel per significante use case een testscript op, waarbij je beschrijft hoe en wat getest moet worden om alle condities in het programma te doorlopen.
   Ontwerp de tests conform de behandelde testspecificatietechnieken.
   - Doe dit voor de systeemtest van de zelfgebouwde functies. Zorg ervoor dat een ander teamlid dan de bouwer van een bepaalde functie de test ontwerpt en uitvoert.

5. Realiseer het systeem.
   Bouw het systeem en voer de programmatest uit. Zorg dat aan de volgende eisen wordt voldaan:
   1. Er moet op ieder moment een database diagram aanwezig zijn, dat een exacte afspiegeling van de tabelstructuur is.
   2. De use cases moeten gebruiksvriendelijk geïmplementeerd worden:
      - Goede navigatie
      - Geen technische sleutels of alleen Id's. En gebruik Lists of Values waar zinvol.
      - Geef goede foutmeldingen die de gebruiker verder helpen.
      - De labels van entiteiten en attributen moeten in alle schermen gelijk zijn.
   3. Het systeem moet volgens de architectuur en het technisch ontwerp zijn gerealiseerd.
      Gebruik HUSACCT voor het Java-deel om de "architectural conformance" te controleren.
   4. De source code moet goed te begrijpen zijn (goede variabelenamen, duidelijke structuur en formatering, no "code smells" etc.).

6. Voer de systeemtest uit.
   - Maak zo nodig de initiële gegevens aan in de database.
   - Voer per testgeval de testdata in. Vergelijk de uitkomst met het verwachtte resultaat en noteer de bevindingen in het testdocument.
   - Pas het systeem zo nodig aan hertest het.

7. Stel het Construction Report op, dat de volgende onderdelen moet bevatten:
   1. Overzicht van de resultaten, werkverdeling en besteedde uren:
      1.1. Welke requirements zijn wel en welke zijn niet gerealiseerd?
      1.2. Wie heeft aan welk onderdeel gewerkt?
      1.3. Hoeveel uur is er per activiteit/persoon besteed (gebruik de meegeleverde spreadsheet)?
   2. Technische documentatie: Implementatie van Software Architectuur & Design Patterns
      2.1. Deelsysteem Define …
         2.1.1. Database structuur diagram van de business rule generator.

2.1.2. Bij implementatie met APEX: Packagestructuur
1) zoals ontworpen in het Architecture notebook;
2) zoals gerealiseerd; met per package een specificatie van de verantwoordelijkheid.
2.1.3. Afwijkingen van de Software Architectuur en het Design.
2.2. Voor ieder deelsysteem (Define … en Transform …) ontwikkeld met Java, aparte uitwerkingen van:
2.2.1. Class diagram met de geïmplementeerde Java-klassen.
2.2.2. Een “Intended Architecture diagram” (HUSACCT) dat de ontworpen modulen (lagen, componenten, …) met dependencies en violations (als die er zijn) laat zien.
2.2.3. Een “Implemented Architecture diagram” (HUSACCT) dat de belangrijkste packages met dependencies en eventuele violations laat zien.
2.2.4. Een specificatie waar welke design patterns zijn toegepast.
Laat dit zien aan de hand van Class diagram(s) en/of Implemented Architecture diagram(s).
2.2.5. Afwijkingen van Software Architectuur en Design.
3. Testresultaten:
3.1. De fouten die tijdens het testen zijn geconstateerd.
3.2. Known bug list: De fouten die in deze versie nog niet hersteld zijn.
4. Evaluatie:
4.1. Verbeterpunten in eigen product en werkwijze.
4.2. Alleen Construction 2: Beoordeling van het project/projectopdracht met pluspunten (wat heb je geleerd, wat was leuk) en minpunten (wat moet er verbeterd worden).

NB Dus lang niet alle producten van de Construction fase hoeven in het Construction Report opgenomen te worden! Het merendeel van de producten van deze fase is voor eigen gebruik en nodig om tot goede en goed geteste resultaten te komen.

8. Presenteer de resultaten in het Engels (circa 25 minuten)
Lever het Construction report op tijd op, conform de planning in Canvas.
1. Demonstreer de functionaliteit in een voor het publiek logische volgorde.
Bedenk een demonstratiescenario en zorg voor goede, realistische data binnen het demonstratiesysteem.
2. Leg uit en laat zien:
2.1. Hoe de software partitioning (uit het architecture notebook) geïmplementeerd is.
2.1.1. APEX: Package ontwerp versus realisatie.
2.1.2. Java: Defined Architecture diagram(s), Implemented Architecture diagram(s)
2.2. Waar welke design patterns zijn toegepast.
2.3. Welke verschillen er tussen architectuur/ontwerp en realisatie zitten.
3. Bespreek kort:
3.1. De belangrijkste fouten die tijdens het testen zijn gevonden.
3.2. De known bug list.
3.3. Werkverdeling.
3.4. Besteedde uren.
3.5. Evaluatie.

Voorbeeld van een demonstratiescenario
1. Leg uit wat het demonstratiesysteem is en toon de databasestructuur daarvan.
2. Laat zien dat de te genereren business rules nog niet in de database staan.
3. Voeg een nieuwe business rule toe.
Laat zien welke controles daarbij worden uitgevoerd.
4. Wijzig een bestaande busines rule.
Laat zien welke (andere) controles daarbij worden uitgevoerd.
5. Selecteer business rules die meegenomen moeten worden voor de generatie.
6. Genereer deze business rules.
7. Voeg de rules toe aan de database van het demonstratiesysteem.
8. Toon de code van de gegenereerde busines rule.
9. Test de rule binnen het demonstratiesysteem.

5. Requirements: Business Rule Generator

5.1 Functional Requirements

1. Business rule typen die data gerelateerd zijn, moeten vastgelegd kunnen worden.
   Van ieder business rule type moet worden geregistreerd: code, naam, toelichting (eventueel met een voorbeeld).
2. Nieuwe business rules moeten kunnen worden vastgelegd en bestaande business rules moeten kunnen worden opgezocht en gewijzigd of verwijderd worden. Het zoeken van business rules moet (in ieder geval) kunnen op basis van naam, tabel en business rule type.
3. Business rules moeten kunnen worden geselecteerd voor code generatie.
4. De geselecteerde set business rules moet conform hun type en definitie exact kunnen worden getransformeerd in de gespecificeerde code en volgens de gespecificeerde naamgevingsconventie.
5. De gegenereerde code moet vanuit het tool aan de database kunnen worden aangeboden en uitgevoerd worden.
6. Zorg ervoor dat bij hergeneratie de eerdere gegenereerde code overschreven/verwijderd wordt.

5.2 Non-functional Requirements

Id	Omschrijving	Kwaliteitsattribuut
NFR1	Het systeem moet eenvoudig te gebruiken en te bedienen zijn.	Gebruikers-vriendelijkheid
NFR2	De ingevoerde rules moeten waar mogelijk op syntax en semantiek gecontroleerd worden.	Functionaliteit/ Juistheid
NFR3	In eerste instantie zal het systeem alleen op een Oracle database werken. Maar het moet later eenvoudig mogelijk zijn om: • De definitie van de business rules ook in andere typen databases op te slaan. • Code voor de rules te genereren voor andere typen databases.	Onderhoudbaarheid en Portabiliteit
NFR4	De twee deelsystemen (Define … en Transform …) moeten onafhankelijk van elkaar vervangen kunnen worden door systemen die met andere technologieën ontwikkeld zijn.	Onderhoudbaarheid
NFR5	Het systeem moet eenvoudig uitbreidbaar zijn met andere typen business rules, zo mogelijk zonder de code aan te passen. Er zijn namelijk nog meer typen data constraints en ook nog hele andere categorieën, zoals authorization rules, die later ook in het tool zouden kunnen worden opgenomen.	Onderhoudbaarheid

5.3 Constraints

1. Het systeem moet licentie-vrij te gebruiken zijn (naast licenties voor database, etc).
2. Het systeem moet als open source opgeleverd en verder ontwikkeld (kunnen) worden.

5.4 De te ondersteunen business rule typen per Construction-iteratie

De volgende typen business rules moeten ondersteund gaan worden (specificatie in het volgende hoofdstuk). Let op de wanneer dit moet gebeuren (welke Construction-iteratie) en de prioriteit.
Iteratie 1 mag zich (hoeft niet) beperken tot alleen de generatie database constraints.
Iteratie 2 moet ook het eenvoudige trigger-mechanisme ondersteunen.

Categorie	Business Rule Type	Iteratie	Prioriteit
Static data constraint rules	Attribute Range rule	1	Must have
	Attribute Compare rule	1	Must have
	Attribute List rule	1	Should have
	Attribute Other rule	1	Could have
	Tuple Compare rule	2	Must have
	Tuple Other rule	2	Could have
	Inter-Entity Compare rule	2	Must have
	Entity Other rule	2	Should have
Dynamic data constraint rules	Modify rule	2	Should have

Naast de bovenstaande typen zijn er nog andere typen business rules, maar die:

- Kunnen met andere tools worden gegenereerd (op basis van een database diagram of een klassendiagram). Zoals regels m.b.t. datatype, optionaliteit, uppercase, uniciteit, foreign key, multipliciteit of update/delete rules bij master/detail-relaties.
- Komen in Iteraties die buiten dit project vallen.
- Of zullen handmatig toegevoegd moeten worden.

Er zijn nog meer categorieën en business rule typen (waar je in dit project niets mee hoeft te doen).
Lauri L. Boyd beschrijft die in de publicatie:
CDM RuleFrame - The Business rule implementation that saves you work
Deze publicatie is op de cursussite te vinden als: CDM Ruleframe - article - Boyd.pdf.

5.5 Naamgeving van database-objecten, foutmeldingen, etc.

Het is wenselijk dat code die uit een generator afkomstig is, makkelijk herkenbaar is voor ontwikkelaars. Daarom is een consistente naamgeving van alles wat door de generator gemaakt wordt (constraints, triggers, foutmeldingen, etc.), noodzakelijk. Zo gebruikt Oracle Designer de aanduiding "CG$", en Headstart/Ruleframe "QMS". Voor de Business Rule Generator kunnen we bijvoorbeeld "BRG" gebruiken.
Andere onderdelen van een naam die de herkenbaarheid bevorderen, zijn:

- de applicatie waar het over gaat (bijvoorbeeld: "VBMG")
- de entiteit waar de rule betrekking op heeft (bijvoorbeeld: "PRT" voor de entiteit "product")
- het gegenereerde object (bijvoorbeeld: "CNS" voor constraint)
- de soort rule (bijvoorbeeld: "TCMP" voor tupel compare rule)
- een volgnummer

Zo zou een check constraint bijvoorbeeld kunnen heten:
BRG_VBMG_PRT_CNS_TCMP_01
De business rule zelf (onafhankelijk van het database object) zou dan BRG_VBMG_PRT_TCMP_03 heten.
Houd bij het samenstellen van zulke codes rekening met de maximale lengte van de namen van veel database-objecten (30 tekens).

6. Specificatie van de Business Rule Typen met voorbeeld-code

Zie het volgende hoofdstuk voor een toelichting bij de code.

Naam	**Attribute Range rule**
Code	ARNG
Categorie	Static data constraint rules
Beschrijving	De attribuutwaarde moet wel of niet overeenkomen met een waarde uit een reeks (range).
Voorbeeld	Tentamen.cijfer mag de waarde 1 tot en met 10 hebben. Alle postcodes mogen meedoen, behalve 3700 tot en met 4100.
Triggering Event	insert, update <<column>> | <<table>>
Trigger-code	l_passed := p_tnt_row.new_cijfer between 1 and 10;
Constraint-statement	cijfer between 1 and 10

Naam	**Attribute Compare rule**
Code	ACMP
Categorie	Static data constraint rules
Beschrijving	De attribuutwaarde moet gelijk, ongelijk, groter, kleiner, ‘groter of gelijk’ of ‘kleiner of gelijk’ zijn aan een opgegeven waarde.
Voorbeeld	Levering.aantal moet groter of gelijk zijn aan 1.
Triggering Event	insert, update <<column>> | <<table>>
Trigger-code	l_passed := p_lev_row.new_aantal >= 1;
Constraint-statement	aantal >= 1

Naam	**Attribute List rule**
Code	ALIS
Categorie	Static data constraint rules
Beschrijving	De attribuutwaarde moet wel of niet overeenkomen met een waarde uit een reeks.
Voorbeeld	De status van een bestelling mag alleen zijn “geregistreerd”, “goedgekeurd”, “magazijn”, “verzonden” of “afgeleverd”.
Triggering Event	insert, update <<column>> | <<table>>
Trigger-code	l_passed := p_xxx_row.new_status in ('geregistreerd', 'goedgekeurd', 'magazijn', 'verzonden', 'afgeleverd')
Constraint-statement	status in ('geregistreerd', 'goedgekeurd', 'magazijn', 'verzonden', 'afgeleverd')

Naam	**Attribute Other rule**
Code	AOTH
Categorie	Static data constraint rules
Beschrijving	De attribuutwaarde moet voldoen aan een beperking die in SQL is gecodeerd.
Voorbeeld	De eerste positie van het huisnummer moet een cijfer zijn.
Triggering Event	insert, update <<column>> | <<table>>
Trigger-code	l_passed := substr(p_adr_row.new_huisnr, 1,1) between ('1' and '9')
Constraint-statement	substr(huisnr, 1,1) between ('1' and '9')

Naam	**Tuple Compare rule**
Code	TCMP
Categorie	Static data constraint rules
Beschrijving	De attribuutwaarde moet gelijk, ongelijk, groter, kleiner, 'groter of gelijk' of 'kleiner of gelijk' zijn aan de waarde van een ander attribuut in het zelfde tuple.
Voorbeeld	Levering.registratieDatum moet kleiner of gelijk zijn aan Levering.leverDatum.
Triggering Event	insert, update << column, column >> | <<table>>
Trigger-code	l_passed := p_lev_row.new_registratiedatum <= p_lev_row.new_leverdatum
Constraint-statement	registratiedatum <= leverdatum

Naam	**Tuple Other rule**
Code	TOTH
Categorie	Static data constraint rules
Beschrijving	De attribuutwaarde moet voldoen aan een beperking die in SQL is gecodeerd en waarbij een of meerdere attributen uit dezelfde tupel zijn betrokken.
Voorbeeld	Als het woonland Nederland is, moet de postcode bestaan uit vier cijfers en twee letters.
Triggering Event	insert, update << column, column >> | <<table>>
Trigger-code	De gebruiker van het tool schrijft een eigen stuk PL/SQL dat l_passed bepaalt. Zie voor een code voorbeeld de andere Other-rules.
Constraint-statement	*meestal niet mogelijk (gebruiker kan eventueel eigen constraint schrijven)*

Naam	**Inter-Entity Compare rule**
Code	ICMP
Categorie	Static data constraint rules
Beschrijving	De attribuutwaarde moet gelijk, ongelijk, groter, kleiner, 'groter of gelijk' of 'kleiner of gelijk' zijn aan de waarde van een ander attribuut in een ander tuple van een andere tabel.
Voorbeeld	Levering.registratieDatum moet groter of gelijk zijn aan Order.registratieDatum.
Triggering Event	insert, update << column, table >> | << column, table>>

Trigger-code voor entiteit Levering	cursor lc_ord is
	select ord.registratiedatum
	from orders ord
	where ord.id = p_lev_row.new_ord_id;
	l_orderdatum orders.registratiedatum%type;
	begin
	open lc_ord;
	fetch lc_ord into l_orderdatum;
	close lc_ord;
	l_passed := p_lev_row.new_registratiedatum >= l_orderdatum;

Trigger-code voor entiteit Order *(n.b.: alleen nodig als Order.registratiedatum wijzigbaar is)*	cursor lc_lev is
	select min(lev.registratiedatum)
	from leveringen lev
	where lev.ord_id = p_ord_row.new_ord_id;
	l_leverdatum leveringen.registratiedatum%type;
	begin
	open lc_lev;
	fetch lc_lev into l_leverdatum;

	close lc_lev; l_passed := p_ord_row.new_registratiedatum <= l_leverdatum;
Constraint-statement	*niet mogelijk*

Naam	**Entity Other rule**
Code	EOTH
Categorie	Static data constraint rules
Beschrijving	De attribuutwaarde moet voldoen aan een beperking die in SQL is gecodeerd en waarbij attributen van andere tupels uit dezelfde tabel zijn betrokken.
Voorbeeld	Er mogen niet meer dan 99 leverregels bij een levering voorkomen.
Triggering Event	insert, update << column, column >> | <<table>>

Trigger-code	l_aantal pls_integer;
	begin
	select count(*)
	into l_aantal
	from leverregels lvr
	where lvr.lev_id = p_lvr_row.new_ord_id;
	l_passed := l_aantal <= 99;

Constraint-statement	*niet mogelijk*

Naam	**Modify rule**
Code	MODI
Categorie	Dynamic data constraint rules
Beschrijving	Als de waarde(n) van een of meerdere attributen in een of meerdere tabellen gewijzigd worden, moet een bepaalde controle plaatsvinden. Bij dynamische regels kun je achteraf niet meer aan de gegevens zien of aan de controle voldaan wordt.
Voorbeeld	Een bestelling mag niet meer gewijzigd worden, als de status = “afgeleverd”. Er mag geen bestelregel meer aan een bestelling worden toegevoegd, nadat de status van de bestelling op “goedgekeurd” is gezet
Triggering Event	insert, update, delete <<table, table>>

Trigger-code	cursor lc_bst is
	select bst.status
	from bestellingen bst
	where bst_id = p_bsr_row.new_bst_id;
	l_status bestelling.status%type;
	begin
	if p_bsr_row.brg_oper = 'INS' then
	open lc_bst;
	fetch lc_bst into l_status;
	close lc_bst;
	l_passed := l_status != 'goedgekeurd';
	end if;

Constraint-statement	*niet mogelijk*

7. Implementatie van business rules

7.1 Implementatiemogelijkheden

In een Oracle database bestaan twee mogelijkheden om business rules af te dwingen: door middel van database constraints of door middel van database triggers. Beiden hebben voor- en nadelen.

database constraints
voordelen:

- worden altijd afgedwongen: de gegevens voldoen per definitie aan de business rule
- eenvoudig te genereren: zeer weinig code nodig

nadelen:

- beperkt in mogelijkheden: zijn maar bij een paar business rules toepasbaar
- niet erg flexibel: niet dynamisch uit te zetten, foutmeldingen komen "een voor een"

database triggers
voordelen:

- alle soorten business rules zijn mogelijk
- flexibel: dynamisch aan en uit te zetten, eigen code mogelijk, foutmeldingen te combineren

nadelen:

- worden alleen gecontroleerd bij een transactie
- veel meer code nodig dan bij constraints

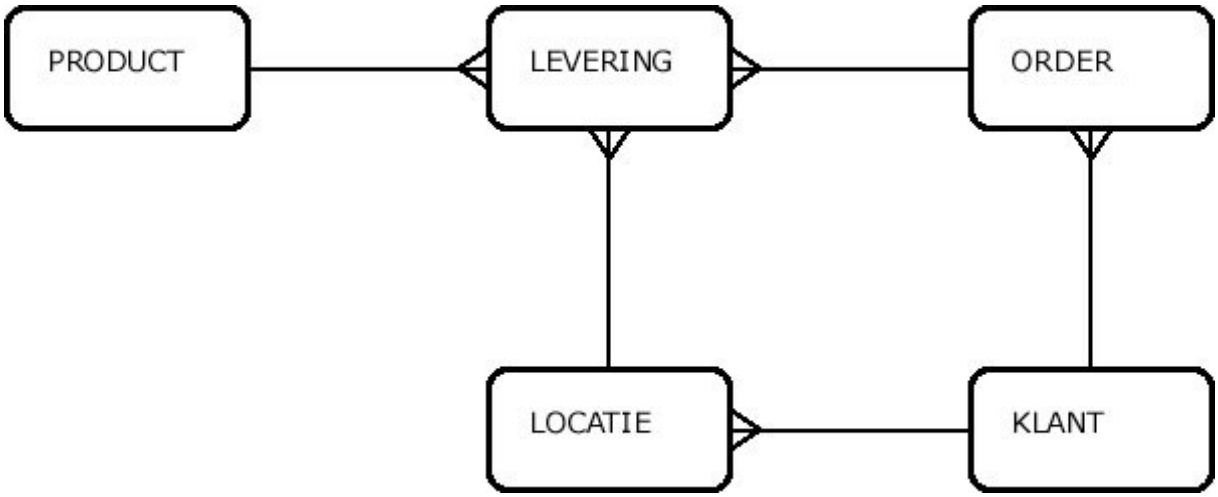
7.2 Toelichting bij de voorbeeld-code

Bij de voorbeelden van code in de specificatie van de Business Rule Typen.
Trigger-code
Hier is steeds een stukje PL/SQL dat een locale variabele l_passed bepaalt. Als l_passed TRUE is, is aan de regel voldaan; als l_passed FALSE is, faalt de regel en moet er een foutmelding verschijnen.
Constraint-statement
Voor business rules die ook met een database constraint geïmplementeerd kunnen worden, wordt ook een deel van het te genereren statement gegeven om de constraint aan te maken. Ieder constraint-statement is als volgt opgebouwd:
alter table <<table>> add constraint <<constraint_name>> check (<<statement>>);
In de voorbeelden is vermeld wat er in <<statement>> moet staan.

8. Codevoorbeelden "Vandaag Besteld Morgen Geleverd"

Op de cursus-site staat de map 'Voorbeelden van Transfer - VBMG database'.
Deze bevat voorbeelden van:
a) constraints;
b) een eenvoudig trigger-mechanisme;
c) een complex trigger-mechanisme.

Schema van de VBMG database: "Vandaag Besteld Morgen Geleverd"



8.1 Voorbeeld-code Eenvoudige Triggeroplossing

Hieronder een implementatie voor een eenvoudige trigger-mechanisme, die geschikt is voor relatief eenvoudige business rules. Paul Koppens heeft in dit voorbeeld drie rules samen in één trigger voor de PRODUCT-tabel ondergebracht.
Ter illustratie heeft hij ook nog een vierde business rule toegevoegd die met deze eenvoudige variant niet gecontroleerd kan worden ("het aantal CD's moet groter zijn dan het aantal boeken"), omdat die een 'mutating table' foutmelding oplevert. Daarvoor is een complex trigger-mechanisme nodig.

Let er op dat de initiële dataset in de tabellen niet aan die business rule voldoet - dat illustreert de beperking dat zo'n rule alleen bij een transactie gecontroleerd wordt.

```
create or replace trigger brg_vbmg_prt_trigger
  before delete or insert or update
  on vbmg_producten
  for each row
declare
  l_oper                varchar2 ( 3 );
  l_error_stack             varchar2 ( 4000 );
begin
  if inserting
  then
    l_oper := 'INS';
  elsif updating
  then
    l_oper := 'UPD';
  elsif deleting
  then
    l_oper := 'DEL';
  end if;
```

```
-- evaluate business rule BRG_VBMG_PRT_ACMP_01
declare
  l_passed              boolean := true;
begin
  if l_oper in ( 'INS', 'UPD' )
  then
    -- de prijs mag niet negatief zijn
    l_passed := :new.prijs >= 0;
    if not l_passed
    then
      l_error_stack := l_error_stack || 'De prijs van het product
            mag niet negatief zijn.';
    end if;
  end if;
end;

-- evaluate business rule BRG_VBMG_PRT_ALIS_01
declare
  l_passed              boolean := true;
begin
  -- het producttype moet BOE, CD of KLE zijn
  if l_oper in ( 'INS', 'UPD' )
  then
    l_passed := :new.product_type in ( 'BOE', 'CD', 'KLE' );
    if not l_passed
    then
      l_error_stack :=
        l_error_stack ||
        'Product type ' ||
        :new.product_type ||
        ' is onbekend. Toegestane waarden zijn BOE, CD en KLE.';
    end if;
  end if;
end;

-- evaluate business rule BRG_VBMG_PRT_MODI_01
declare
  l_passed              boolean := true;
begin
  -- de prijs van een boek mag niet verhoogd worden
  if :new.product_type = 'BOE' and
    l_oper = 'UPD'
  then
    l_passed := :new.prijs <= :old.prijs;
  end if;
  if not l_passed
  then
    l_error_stack := l_error_stack || 'De prijs van een boek mag
        niet verhoogd worden.';
  end if;
end;
```

```
  /* ↓KAN NIET MET DEZE METHODE: LEVERT EEN MUTATING TABLE FOUT OP
    -- evaluate business rule BRG_VBMG_PRT_EOTH_01
    declare
      l_passed              boolean := true;
      l_boek               pls_integer;
      l_cd                 pls_integer;
    begin
      -- er moeten meer CD's dan boeken in de productenlijst staan
      if :new.product_type = 'BOE' and
        l_oper in ( 'INS', 'UPD' ) or
        :old.product_type = 'CD' and
        l_oper in ( 'DEL', 'UPD' )
      then
        select count ( * )
        into l_boek
        from vbmg_producten
        where product_type = 'BOE';
        select count ( * )
        into l_cd
        from vbmg_producten
        where product_type = 'CD';
        l_passed := l_cd > l_boek;
      end if;
      if not l_passed
      then
        l_error_stack := l_error_stack || 'Er moeten meer CD''s dan
              boeken in de productenlijst staan.';
      end if;
    end;
  */
  if l_error_stack is not null
  then
    raise_application_error ( -20800, l_error_stack );
  end if;
end brg_vbmg_prt_trigger;
/
```

8.2 Code voor het complexe database-trigger mechanisme

Om gebruik te kunnen maken van database-triggers die alle typen business rules ondersteunen, zijn voor iedere tabel de volgende objecten nodig:

- een before statement trigger
- een before each row trigger
- een after statement trigger
- een database package

Het database package bevat de eigenlijke logica en ook de specifieke te genereren code voor iedere afzonderlijke business rule. Van de code voor het database package zelf en voor de triggers is een voorbeeld beschikbaar bij de VBMG-voorbeeld database van Transfer Solutions.