Informatiesystemen 1

Thom Wiggers
s4119444

6 januari 2013

Inhoudsopgave

1 Huishoudelijke mededelingen

Dit project maak ik individueel.

2 Taak 1 - ORC vs. SQL

2.1 Inleiding

ORM is een methode om door middel van modellen systemen te ontwikkelen waarbij gepoogd wordt zo min mogelijk fouten toe te laten en zo veel mogelijk redundantie in de opgeslagen gegevens te voorkomen.

ORM bestaat voornamelijk uit een verzameling afspraken over taalgebruik en notaties. Hierdoor zou een goed model ook voor niet-domeinexperts leesbaar en begrijpbaar moeten zijn.

Hoewel ORM modellen vooral worden omgezet naar klassieke relationele (SQL)-databases, is het ook mogelijk om direct 'vragen' te stellen aan een dataset in ORM. Deze querytaal staat bekend als Object-Role Calculus (ORC).

Ik ga hier proberen ORC te vergelijken met de SQL taal, door middel van het vergelijken van enkele verschillende queries, zoekvragen, waarbij ik ook in ga op de fundamentele verschillen tussen de twee verschillende systemen. Hiervoor zal ik een voorbeeldsysteem beschrijven, zowel uitgevoerd in ORM als draaiende op de populaire relationele database postgreSQL.

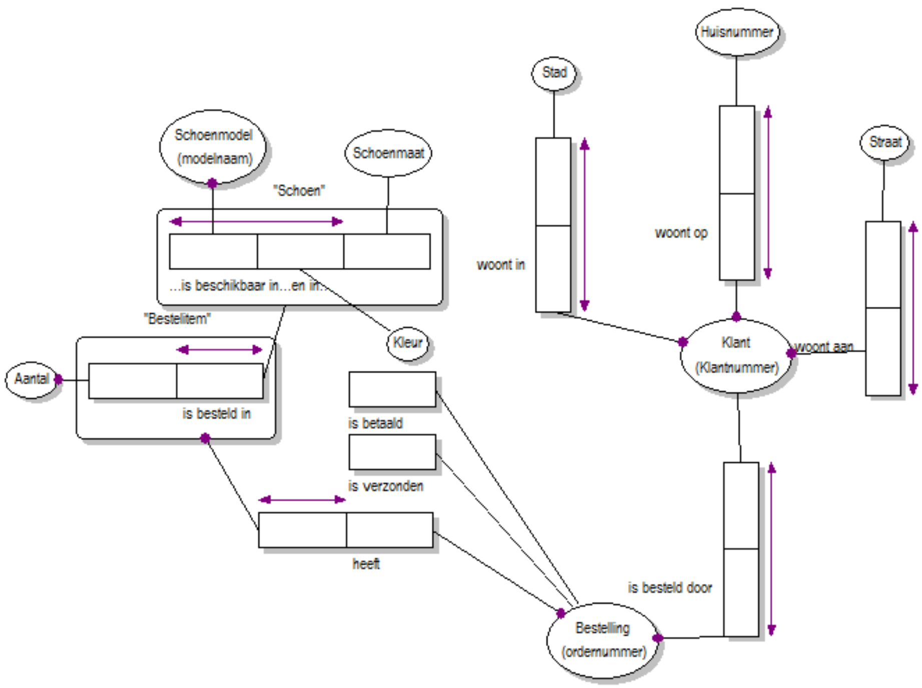
2.2 Systeem

Ik ga hier een voorbeeldsysteem beschrijven van een webwinkel waar men schoenen verkoopt. In deze webwinkel houdt men bestellingen bij, en profielen van klanten. Bestellingen kunnen bestaan uit een of meerdere paren schoenen, in verschillende maten en aantallen.

2.2.1 ORM

— Check syntax — Todo: Fix diagram: Naam, constraint Schoen

```
Schoen: Schoenmodel (modelnaam) is beschikbaar
    in Schoenmaat en is beschikbaar in Kleur.
Bestelitem: Schoen is besteld in aantal.
```



Figuur 1: ORM Model van het voorbeeldsysteem

```
Bestelling (ordernummer) heeft Bestelitem.
Bestelling (ordernummer) is besteld door
   Klant (klantnummer).
Bestelling is betaald.
Bestelling is verzonden.
Klant (klantnummer) woont aan
   Straat (straatnaam).
Klant (klantnummer) woont op
   Huisnummer (nr).
Klant (klantnummer) woont in Stad (naam).
Klant (klantnummer) heet Naam.
```

2.2.2 SQL Tabellen

Het transformeren van het ORM model naar SQL tabellen is redelijk eenvoudig, maar om dubbel voorkomende gegevens te voorkomen heb ik op verschillende plaatsen een identificatienummer aan tabellen toegevoegd als vervangende primary key.

Hier wordt al een duidelijk nadeel van SQL-gebaseerde databases zichtbaar: het is niet mogelijk om gegevens te objectiviseren.

Zie hiervoor de tabellen 1, 2, 3, 4 en 5.

id	**Schoenmodel**	**Kleur**	**Schoenmaat**
1	Model a	Zwart	43
2	Model a	Zwart	42
3	Model a	Wit	43
...	...	...	...

Tabel 1: Schoen tabel

id	**schoenid**	**Aantal**
1	2	1
2	2	2
3	3	1
...	...	...

Tabel 2: Bestelitem Tabel

Bestelnummer	**Besteldddoor**	**Verzonden**	**Betaald**
1	1	Y	Y
2	1	N	Y
3	2	Y	N
...	...	...	...

Tabel 3: Bestellingen

Bestelitem	**Bestelling**
1	1
2	1
3	2
...	...

Tabel 4: BestelitemBestelling: Bestelde items horende bij bestellingen

Klantnummer	**Naam**	**Straat**	**Huisnummer**	**Stad**
1	John Doe	Heyendaalseweg	91	Nijmegen
2	Steve Foo	Asselsestraat	34	Apeldoorn
3	Jane Bar	Kanaalstraat	33	Amsterdam
...	...	...	...	...

Tabel 5: Klant tabel

2.3 Eenvoudige gegevens uit het systeem halen

2.3.1 SQL

Eenvoudige gegevens uit het informatiesysteem halen is eenvoudig in SQL. Een `SELECT` statement is erg eenvoudig voor elkaar te krijgen. Bijvoorbeeld het selecteren van alle verschillende schoenen die in de winkel te koop worden aangeboden:

```
SELECT Schoenmodel, Kleur, Schoenmaat
FROM Schoen;
```

2.3.2 ORC

In ORC is het ook eenvoudig om dezelfde gegevens op te halen:

```
Schoenmodel, Kleur, Schoenmaat FROM Schoen
```

Het verschil tussen ORC en SQL is hier niet zo groot. SQL heeft `SELECT`, maar verder zou men bijna copy/paste kunnen doen.

2.4 Conditioneel gegevens uit het systeem halen

Men wil niet altijd alle gegevens uit een informatiesysteem hebben. Daarom is het in SQL en in ORC mogelijk om condities op te geven waaraan de op te vragen informatie moet voldoen.

2.4.1 SQL

Stel, ik wil alle verschillende modellen van zwarte schoenen hebben in maat 43.

```
SELECT Schoenmodel
FROM Schoen
WHERE Schoenmaat = '43'
  AND Kleur='Zwart';
```

2.4.2 ORC

In ORC gaat het zo:

```
Schoenmodel FROM Schoen in Kleur 'Zwart'
  AND in Schoenmaat '43'
```

Ook hier is er een grote overeenkomst tussen SQL en ORC. SQL heeft hier het `WHERE` keyword om onderscheid te maken tussen de condities en de tabel, maar verder is het grotendeels hetzelfde.

2.5 Complexe informatie uit het systeem halen

Stel, ik wil weten uit welke steden de mensen komen die zwarte schoenen besteld hebben.

2.5.1 SQL

```
SELECT stad
FROM Klant
WHERE Klant.Klantnummer IN
  (SELECT Besteldoor
   FROM   Bestellingen
   JOIN   BestelitemBestelling
   ON   Bestelling = Bestelnummer
   WHERE  Bestelitem IN
     (SELECT id
      FROM Bestelitem
```

```
      WHERE Schoenid IN
        (SELECT id
         FROM Schoen
         WHERE Kleur = 'Zwart')
     )
  )
```

Dit is duidelijk een erg complexe query, omdat men vele tabellen door moet omdat de koppeling tussen de informatie vrij zwak is in Relationele Databases.

2.5.2 ORC

```
Stad woonplaats van Klant
  die besteld heeft Bestelling
    die Bestelitem heeft
    met Schoen in kleur Zwart
```

De ORC query is hier duidelijk een stuk eenvoudiger. Dit komt doordat in een ORM-schema de relaties tussen verschillende gegevens een stuk duidelijker is edefineerd. Waar SQL steeds in verschillende lijsten moet zoeken die allemaal apart gemaakt dienen te worden, is het hier iets wat impliciet gebeurt.

2.6 Conclusie

Waar ORC en SQL vaak erg op elkaar lijken, is ORC soms in staat om veel duidelijker queries op te stellen, vooral wanneer het om complexe informatie gaat. Relationele Databases zijn minder sterk in het weergeven van de verbanden tussen informatie, en er moeten vaak extra gegevenstypen geintroduceerd worden (identificatienummers bijvoorbeeld) om dataduplicatie te voorkomen.

3 Taak 2 - Typegerelateerdheid

3.1 Inleiding

Omdat ik denk dat dit een belangrijk onderdeel is van de stof, en veel van de problemen behandeld in het college een oplossing hebben die steeds subtiel is en niet per se direct voor de hand ligt, ga ik hier proberen een paar extra voorbeelden te maken en uit te werken, om zo mijn begrip van typegerelateerdheid wat scherper te krijgen.

Ik ga proberen om de afleidingsregels zoals vermeld in het dictaat [1, p. 41] opnieuw af te leiden.

3.2 Regels

3.2.1 T1

$$\vdash x \sim x$$

waar $\sim$ typegerelateerd betekent.

Type x is vanzelfsprekend typegerelateerd met zichzelf. De typegerelateerdheidsrelatie is *reflexief*.

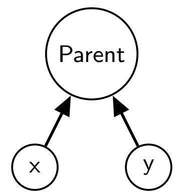
3.2.2 T2

$$x \sim y \vdash y \sim x$$

Typegerelateerdheid is ook een *symmetrische* relatie.

3.2.3 T3

$$\sqcap(x) = \sqcap(y) \wedge y \sim z \vdash x \sim z$$



Figuur 2: Specialisatie

waar $\sqcap(x)$ de *Pater familias* van x is.

Dit betekent dat als x en y specialisaties van hetzelfde type zijn, en y is typegerelateerd aan z, x ook typegerelateerd is aan z. Zie hiervoor ook figuur 2. Elementen in "Parent" ($\sqcap(x)$ en $\sqcap(y)$) komen ook voor in x en in y, daar zij specialisaties van "Parent" zijn, ze hebben grotendeels dezelfde eigenschappen, alleen een paar meer. Elementen die typegerelateerd zijn aan "Parent" en aan één van de subtypes van "Parent" zijn dus ook typegerelateerd aan de andere subtypes.

Er is ook het bijzondere geval waarbij $y = z$, waarmee bewezen kan worden dat $x \sim y$.

3.2.4 T4

$$x \,\mathrm{Gen}\, y \wedge y \sim z \vdash x \sim z$$

waar $x \,\mathrm{Gen}\, y$ betekent "x is een generalisatie van y".

Op dezelfde manier als bij T3 (3.2.3), komen elementen van generalisaties voor in het gegeneraliseerde type. Typen die gerelateerd zijn aan een bepaald type, zijn dus ook gerelateerd aan de generalisaties van dat type.

Een variant van deze waarbij $y = z$ bewijst dat $x \,\mathrm{Gen}\, y \vdash x \sim y$.

3.2.5 T5

$$x, y \in \mathcal{G} \wedge \mathrm{Elt}(x) \sim \mathrm{Elt}(y) \vdash x \sim y$$

Als x en y powertypen zijn, en hun elementen typegerelateerd, dan zijn deze powertypen, verzamelingen van hun elementtypen, natuurlijk ook gerelateerd.

3.2.6 T6

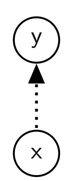
$$x, y \in \mathcal{S} \wedge \mathrm{Elt}(x) \sim \mathrm{Elt}(y) \vdash x \sim y$$

Zoals bij T5 (3.2.5), zijn als de elementen van sequentietypen typegerelateerd zijn, de sequentietypen zelf ook typegerelateerd.

3.2.7 T7

$$\mathcal{O}_x = \mathcal{O}_y \vdash x \sim y$$

Als het objecttype van x en die van y dezelfde zijn, zijn x en y typegerelateerd.



Figuur 3: Generalisatie

3.3 Voorbeeld

Ik gebruik hier [1, figuur 2.26] uit het dictaat. Ik ga de bewering dat alle typen behalve C en B, C en E, en C en F, typegerelateerd zijn toetsen.

$$\begin{aligned}
A \,\mathrm{Gen}\, B &\vdash A \sim B \\
A \,\mathrm{Gen}\, C &\vdash A \sim C \\
\sqcap(D) = \sqcap(A) \wedge A \sim C &\vdash D \sim C \\
\sqcap(D) = \sqcap(A) \wedge A \sim B &\vdash D \sim B \\
\sqcap(A) = \sqcap(D) \wedge D \sim D &\vdash A \sim D \\
B, E \in \mathcal{G} \wedge A \sim D &\vdash B \sim E \\
F, B \in \mathcal{G} \wedge A \sim A &\vdash F \sim B \qquad (B \in A) \\
\sqcap(D) = \sqcap(E) \wedge D \sim A &\vdash E \sim A \\
F, B \in \mathcal{G} \wedge E \sim A &\vdash F \sim A \\
\sqcap(D) = \sqcap(E) \wedge D \sim D &\vdash D \sim E \\
E, F \in \mathcal{G} \wedge D \sim E &\vdash E \sim F \\
\sqcap(D) = \sqcap(F) \wedge D \sim A &\vdash D \sim F
\end{aligned}$$

3.4 Conclusie

Typegerelateerdheid is iets wat op een redelijk intuïtief vlak werkt, maar ook een aantal harde regels kent. Veel van die regels zijn makkelijk beredeneerbaar, zoals bijvoorbeeld typegerelateerdheid van powertypen, en sommige anderen zijn iets moeilijker om te onderbouwen. Het consequent toepassen van die regels is echter vaak een flinke uitdaging, en waar het vaak makkelijk is om te zeggen "x is gerelateerd aan y, want zus en zo", is dit opschrijven volgens de bovengenoemde regels (3.2) vaak een uitdaging.

4 Taak 3 - Constraints en populeerbaarheid

4.1 Inleiding

In deze taak probeer ik wat ingewikkelder contraints en restricties op domeinmodellen te vinden, en daar wat voorbeelden bij te maken en uit te werken. In het bijzonder ga ik het deze keer proberen op te schrijven zonder gebruik te maken van afbeeldingen[1]. Ik ga ook proberen verschillende constraints in een model te gebruiken en kijk naar de populeerbaarheid.

[1] Ik maak wel gebruik van schema's in mijn notities, omdat het bedenken van een kloppend schema door willkeurig opschrijven van symbolen niet echt doenbaar is.

4.2 Systeem 1

$$
\begin{aligned}
\mathcal{P} &= \{p1, p2, p3, p4\} \\
\mathcal{E} &= \{A, B, C\} \\
\mathcal{F} &= \{f, g\} \\
\mathcal{G} &= \emptyset \\
\mathcal{S} &= \emptyset \\
\text{Spec} &\Rightarrow B \,\text{Spec}\, A \\
f &= \{p1, p2\} \\
g &= \{p3, p4\} \\
\text{Base}(p1) &= A \\
\text{Base}(p2) &= B \\
\text{Base}(p3) &= A \\
\text{Base}(p4) &= C \\
\text{total}(\tau) &\text{ where } \tau = \{p1\} \\
\text{total}(\tau) &\text{ where } \tau = \{p2\}
\end{aligned}
$$

Instanties van A moeten in f een rol spelen (total($p1$)) en ook alle elementen in B moeten dat (total($p2$)). B is daarnaast een specialisatie van A, met als gevolg dat instanties van B ook in A voorkomen. Dit betekent echter niet dat de populaties van A en B gelijk moeten zijn: B kan een subset zijn van de instanties van A. Een voorbeeld zou zijn dat A studenten betekent, en B studentmentoren.

Dit systeem is Globaal Objectpopuleerbaar: want er is een voorbeeld populatie:

$$
\begin{aligned}
\mathrm{Pop}(A) &= \{a1, a2\} \\
\mathrm{Pop}(B) &= \{a2\} \\
\mathrm{Pop}(C) &= \{c\} \\
\mathrm{Pop}(f) &= \{\{a1, a2\}, \{a2, a2\}\} \\
\mathrm{Pop}(g) &= \{\{a1, c\}\}
\end{aligned}
$$

Doordat dit model Globaal Objectpopuleerbaar is, is het direct ook Globaal Atomair populeerbaar, Lokaal Atomair populeerbaar, en Lokaal Objectpopuleerbaar. Voor populaties in de bovenstaande klassen kan men hetzelfde voorbeeld gebruiken.

Stel we zouden de constraint unique($\{p1\}$)$\wedge$unique($\{p2\}$) toevoegen. Dan zou er een 1-op-1 mapping van A op B ontstaan, waaraan door de totale rol constraints alle elementen van A en B mee moeten doen. Dan creëert men dus een situatie waarin $\mathrm{Pop}(A) = \mathrm{Pop}(B)$.

Als men in dit systeem total($p1$) zou weglaten, en unique($\{p2, p4\}$) toevoegt, dan ontstaat een leuk nieuw systeem, waarin men $\xi(\{f, g\})$ kan doen.

$$\xi(\tau) = \sigma_{p1=p3}(f \bowtie g)$$

Hierdoor krijgt men het volgende systeem:

$$
\begin{aligned}
\mathcal{P} &= \{p1, p2, p4\} \\
\mathcal{E} &= \{A, B, C\} \\
\mathcal{F} &= \{f \bowtie g\} \\
\text{Spec} &\Rightarrow B\,\text{Spec}\,A \\
f \bowtie g &= \{p1, p2, p4\} \\
\text{Base}(p1) &= A \\
\text{Base}(p2) &= B \\
\text{Base}(p3) &= C \\
\text{unique}(\tau) &\text{ where } \tau = \{p2, p4\}
\end{aligned}
$$

Uiteraard is dit systeem Globaal Objectpopuleerbaar:

$$\text{Pop}(A) = \{a\},\ \text{Pop}(B) = \{b\},\ \text{Pop}(C) = \{c\}$$

4.3 Systeem 2

In dit onderdeel gebruik ik het volgende systeem:

$$
\begin{aligned}
\mathcal{P} &= \emptyset \\
\mathcal{E} &= \{A, B, C\} \\
\mathcal{F} &= \emptyset \\
\mathcal{G} &= \{A, C\} \\
\mathcal{S} &= \emptyset \\
\mathrm{Spec} &\Rightarrow \mathcal{G}(A)\,\mathrm{Spec}\,C \\
&\Rightarrow B\,\mathrm{Spec}\,A \\
\mathrm{Gen} &\Rightarrow C\,\mathrm{Gen}\,B \\
\mathrm{total}(\tau) &\text{ where } \tau = \{\mathcal{G}(C)\}
\end{aligned}
$$

Referenties

[1] Advanced Information Models, Arthur ten Hofstede en Patrick van Bommel, Radboud Universiteit Nijmegen, September 2011.