# Universiteit Antwerpen

Universitaire Instelling Antwerpen
Departement Wiskunde-Informatica


## Temporele Entity-Relationship modellering en aspecten van temporele databanken, toegepast op de historische databank van het Studiecentrum voor Onderneming en Beurs


Promotor – prof. dr. Jan Paredaens
Copromotor – dr. Marc Gemis
Begeleider – dr. Frans Buelens

Academiejaar – 1998-1999


Thesis aangeboden door
Claassen Arvid

tot het bekomen van de graad
Licentiaat in de Wetenschappen

## Woord van dank

Een thesis schrijven doet niemand alleen, daarom wil ik aan het begin van dit document alle mensen bedanken die –direct en indirect– betrokken waren bij het opzoekingswerk voor en het opstellen van mijn thesis.

Eerst en vooral zijn dat natuurlijk mijn promotor prof. dr. Jan Paredaens, co-promotor dr. Marc Gemis en begeleider dr. Frans Buelens, zij gaven mij het afgelopen jaar het gevoel dat ik met hen samenwerkte in plaats van dat zij mijn vorderingen superviseerden. Ik kreeg van hen een enorme vrijheid in mijn doen en laten omtrend deze thesis en het bijbehorende programmeerwerk.

Ten tweede zijn er al die mensen die mij tijdens het afgelopen jaar aan het nodige werkmateriaal geholpen hebben, zowel praktisch als theoretisch: prof. dr. Jan Annaert (univ. Rotterdam), Bart Bemindt (UIA-student), Paul Brown (Informix GB), Bart De Backer (Oracle België), prof. dr. Marc De Ceuster (UFSIA), Stijn Dekeyser (UIA), David De Ridder (UIA-student), Joeri Leemans (UIA-student), Christine Shannon (Informix VS), Karel Sohl (Oracle België), Luc Van de Velde (Informix België), Hans Van Hauteghem (UIA-student), Bart Vossen (Oracle België), Jef Wijsen (UIA), Hans Willems (SCOB) en alle personen die ik ben tegengekomen tijdens mijn urenlange zoektochten op het Web en in nieuwsgroepen. Deze laatsten hebben me vooral geholpen bij het controleren, bevestigen of verwerpen van informatie die ik bekomen heb van de grote commerciële bedrijven.

Tot slot maar zeker niet in mindere mate wil ik mijn ouders bedanken die mij de kans hebben gegeven om mijn informaticastudies te beginnen en af te maken. Ze hebben me het afgelopen jaar mijn gang laten gaan en hebben me, zonder te storen, laten werken aan dit document, ook al was dat niet altijd even haalbaar. In het bijzonder wil ik mijn moeder danken voor de vele uren die zij gespendeerd heeft aan de correctie van de spellingsfouten in dit document, ondertussen moet zij een kenner geworden zijn op het gebied van temporele databanken.

Dankuwel !!!

Arvid Claassen
14 juni 1999

# Inleiding

Deze thesis maakt deel uit van het project van het *Studiecentrum voor Onderneming en Beurs* (afgekort SCOB). Het SCOB heeft tot doel het archief van de Beurs van Brussel (in het bezit van RUCA) toegankelijk te maken door enerzijds de bedrijfshistorische informatie op een correcte manier te inventariseren en anderzijds de koersinformatie te digitaliseren. Beide delen van het archief zullen met elkaar verbonden worden, zodat op eenvoudige manier fundamentele informatie uit het bedrijfshistorische gedeelte gekoppeld kan worden aan relevante koers- en rendementsgegevens die afgeleid worden uit de koerslijsten.
Het project beoogt over te gaan tot praktisch onderzoek. De financieel-economen zullen de beschikbaarheid van lange tijdreeksen gebruiken om na te gaan hoe de aandeelkoersen statistisch verdeeld zijn en in welke mate deze verdeling stabiel blijft in de tijd. Met name het zich voordoen van economisch turbulente periodes kan een belangrijke impact hebben op de statistische beschrijving van aandeelkoersen en -rendementen.

De werkgroep informatica van het SCOB heeft de taak om de archiefstructuur te modelleren in het relationele model en met dat model een databank op te starten. Tegelijk moet de nodige programmatuur ontwikkeld worden om de koersgegevens te kunnen invoeren. Het relationele model is reeds zo goed als volledig uitgewerkt door dr. Gemis in het SCOB-werkdocument «*Ontwerp van een databank m.b.t. het archief van de beurs van Brussel*» (afgekort: [GEMI98]) waarin bijna 150 relationele tabellen zijn opgenomen tezamen met een begeleidende tekst; [GEMI98] vormt dan ook een stevige basis voor dit thesisdocument.

Deze thesis is opgedeeld in een theoretisch en een praktisch deel die elkaar meermaals overlappen. Beide delen zijn verder opgedeeld in thematische hoofdstukken. In het theoretische gedeelte worden ten eerste verschillende temporele *entity-relationship*-modellen naast elkaar gelegd en vergeleken. Daarnaast wordt ook een gelaagd ER-model bekeken. Aan de hand van deze modellen wordt dan een gelaagd, temporeel model voorgesteld. Ten derde worden enkele belangrijke aspecten van temporele databanken beschouwd, zoals temporele duplicaten, intervalmaximalisatie en temporele vraagtalen. Bij de uitwerking van deze onderwerpen wordt de toepasbaarheid ervan op het SCOB-project nooit uit het oog verloren.

Het praktische gedeelte omhelst voornamelijk de ontwikkeling van een Javamodule die de gebruiker in staat stelt om op een vlotte manier relevante koersgegevens in te voeren in de SCOB-databank, aan de hand van de beschikbare informatie uit het archief. Minder intensief maar daarom zeker niet minder belangrijk zijn: de toepassing van het in de theorie ontwikkelde gelaagde en temporele ER-model op het model uit het SCOB-werkdocument en het opstellen van een toekomstplanning voor de werkgroep informatica. Tot slot van deze thesis wordt dan nog een overzicht gegeven van de belangrijkste commerciële databanksystemen.

De digitale versie van dit document (opgesteld in Microsoft® Word 97) en de volledige Javacode van de ontwikkelde programmatuur zullen vanaf 15 juni 1999 beschikbaar zijn op volgende Internetadres:
http://win-www.uia.ac.be/u/s970060/thesis.zip en dit nog tot zeker september 1999.

Voor vragen ben ik bereikbaar op volgende coördinaten:

Claassen Arvid
Dorpsstraat 17
2070 Burcht
tel./fax: 03/252.91.79
arvid.claassen@uia.ac.be of nibblus@hotmail.com

# Hoofdstuk 1 : Tijdreeksen

Vooraleer temporele ER-modellen onder de loep te nemen, wordt eerst het tijdreekstype besproken. Dit is als volgt te verantwoorden: het tijdreekstype zal veelvuldig gebruikt worden in de uiteindelijke SCOB-databank, bijgevolg is het aangeraden om bij het modelleren van de databank een techniek te gebruiken die expliciet in staat is om tijdreeksen te modelleren naast minder gestructureerde temporele aspecten. Bij de bespreking en beoordeling van temporele ER-modellen zal het expliciet aanwezig zijn van tijdreeksen dan ook een belangrijk criterium zijn.
In dit hoofdstuk wordt het tijdreekstype gepresenteerd op een tamelijk hoog niveau, waarbij de onderliggende implementatie van het type door een DBMS-producent buiten beschouwing wordt gelaten.

## *1.1 Definitie*

Een tijdreeks is een rij observatiewaarden, gegenereerd over een bepaalde periode en kan worden voorgesteld als een geordende verzameling koppels:

*Tijdreeks(W)={($t_1$,$w_1$), ($t_2$,$w_2$), … ($t_n$,$w_n$)}*

$t_i$ is het tijdstip waarop de observatie $w_i$ plaatsvond
*W* is het domein van de observatiewaarden

Een tijdreeks heeft volgende eigenschappen:

- De observaties moeten gebeuren op geregelde tijdstippen dus $\Delta t=t_{i+1}-t_i$ moet constant zijn. Deze eigenschap kan omzeild worden door onregelmatige metingen op regelmatige tijdstippen te interpolleren. Dit heeft uiteraard enkel zin bij tijdreeksen waarvan de interpollatie een benadering is van de geobserveerde werkelijkheid.
- De geobserveerde waarden zijn numeriek dus $W \subseteq \Re$.
- De informatie van een tijdreeks bevindt zich voornamelijk in de vorm van de tijdeeks, dus in het verloop van de waarden in de tijd; de individuele observatiewaarden zijn van minder belang.
- Op elk moment $t_i$ kunnen meerdere gegevens tegelijk geobserveerd worden; een tijdreeks wordt dan als volgt voorgesteld:

  *Tijdreeks($W_1$, …, $W_m$)={($t_1$,$w_{11}$,…,$w_{1m}$), …, ($t_n$, $w_{n1}$, …, $w_{nm}$)}*

## *1.2 Tijdsafhankelijke attributen in het relationele model*

Louter relationeel beschouwd zijn tijdreeksen niets nieuws want een tijdreeks kan relationeel gemodelleerd worden als een meerwaardig attribuut waarbij elke attribuutwaarde geïndexeerd wordt met een tijdstip.

Voorbeeld: Een aandeel heeft een naam, een emittent (d.i. het bedrijf dat het aandeel uitgeeft) en dagelijkse koersen. Een koers kan worden voorgesteld door een meerwaardig samengesteld attribuut *Koers(datum, bedrag)*. Ongenormaliseerd ziet de relationele tabel er als volgt uit:

| Sleutel | Naam | Emittent | Koers |
|---|---|---|---|
| 1 | SCOB | UA | (2 jan 1999, 456)<br>(4 jan 1999, 462)<br>(5 jan 1999, 464)<br>… |
| 2 | Archief | BvB | (2 jan 1999, 124)<br>(4 jan 1999, 124)<br>(5 jan 1999, 125)<br>… |
| 3 | … | … | … |

**Tabel 1:** ongenormaliseerde tabel met een tijdreeks.

Bij de normalisering wordt het meerwaardige attribuut *Koers* afgesplitst en in een nieuwe tabel opgenomen. In de eerste normaalvorm is tabel 1 omgezet in Tabel 2 en Tabel 3. Tabel 2 bevat alle tijdsonafhankelijke attributen van een aandeel, Tabel 3 bevat de genormaliseerde tijdreeks. De sleutelattributen zijn hier (*AandeelSleutel*, *Tijdstip*), waarbij *AandeelSleutel* een refererende sleutel is naar het sleutelattribuut *Sleutel* in Tabel 2.

| Sleutel | Naam | Emittent |
|---|---|---|
| 1 | SCOB | UA |
| 2 | Archief | BvB |
| 3 | … | … |
| … | | |

**Tabel 2:** overblijvende tijdsonafhankelijke attributen uit Tabel 1.

| AandeelSleutel | Tijdstip | Koers |
|---|---|---|
| 1 | 2 jan 1999 | 456 |
| 1 | 4 jan 1999 | 462 |
| 1 | 5 jan 1999 | 464 |
| 2 | 2 jan 1999 | 124 |
| 2 | 4 jan 1999 | 124 |
| 2 | 5 jan 1999 | 125 |
| … | | |

**Tabel 3:** eerste normaalvorm van een tijdreeks.

Om in de genormaliseerde tabellen een analyse uit te voeren op de koersgegevens van het aandeel *SCOB*, moeten volgende stappen worden uitgevoerd:

- Zoek in Tabel 2 de sleutelwaarde van het aandeel met de naam *SCOB*.
- Selecteer uit Tabel 3 alle tuppels met als *Aandeelsleutel*-waarde de gevonden sleutelwaarde uit de eerste stap.
- Orden de gevonden tuppels volgens *Tijdstip*. Het resultaat van deze stap is de *Koers*-tijdreeks voor het aandeel *SCOB*.
- Voer de gewenste analyse uit op de resultaattuppels.

Strikt relationeel kan een tijdreeks nauwelijks anders voorgesteld worden dan volgens de uiteengezette methode. Dit is niet echt performant omdat er geen expliciet gebruik wordt gemaakt van bepaalde eigenschappen van tijdreeksen; zo kan een tijdreeks een voorspelbaar patroon vertonen. Bv. koersen worden enkel op weekdagen genoteerd en er is altijd juist één koerswaarde per dag. In dit geval is het overbodig om voor elke dag een *Tijdstip*-waarde te stockeren, want als één tijdstip gegeven is dan kan aan de hand van het patroon (in dit geval weekdagen) uitgerekend worden wat het volgende tijdstip is.
Om een analyse te kunnen uitvoeren, is ook steeds een sortering nodig volgens *Tijdstip*, anders kan niet gegarandeerd worden dat de elementen in de resultaattabel in de juiste volgorde staan.

Het relationele model bevat tevens geen operaties die bepaalde bewerkingen op tijdreeksen mogelijk maken, m.a.w. de functionaliteiten van een tijdreeks beperken zich tot de SQL-operatoren. Een relationele tijdreeks mist ook aan algemeenheid: de modelleerder moet voor elke tijdreeks manueel de extra tabel aanmaken.

## *1.3 Het datatype Tijdreeks*

De belangrijkste databankproducenten[1] hebben ingezien dat er nood is aan een expliciet tijdreekstype en hebben een abstract datatype (ADT) toegevoegd aan hun DBMS. Syntactisch komt het erop neer dat alle gegevens van één tijdreeks worden ingevoerd in één cel van een relationele tabel, zoals in Tabel 1. Die ene cel bevat dus alle tijdsafhankelijke waarden van een (eventueel samengesteld) attribuut van één entiteitsinstantie, ook na normalisering. Hoe de inhoud van de cel in de databank wordt opgeslagen, wordt volledig overgelaten aan het DBMS.

Wanneer een *Tijdreeks*-instantie wordt gebruikt in het relationele model, moet een tijdreeks niet meer worden voorgesteld als een samengesteld meerwaardig attribuut, bijgevolg ontstaan er door het gebruik van tijdreeksen geen extra tabellen meer. Tabel 1 ziet er nu uit als Tabel 4.

[1] In hoofdstuk 8 wordt een overzicht gegeven van vijf object-relationele databanken.

| Sleutel | Naam | Emittent | Koers |
|---|---|---|---|
| 1 | SCOB | UA | Tijdreeks(Bedrag) |
| 2 | Archief | BvB | Tijdreeks(Bedrag) |
| 3 | … | … | … |
| … | | | |

**Tabel 4:** tijdreeksen in het relationele model met het *Tijdreeks*-type.

De notatie *Tijdreeks(Bedrag)* in de kolom *Koers* is slechts schematisch: de onderliggende structuur van een cel uit deze kolom wordt afgeschermd van de gebruiker.
Tezamen met het type *Tijdreeks* worden ook een reeks nieuwe operaties toegevoegd aan het DBMS waardoor specifieke bewerkingen en analyses op *Tijdreeks*-instanties mogelijk worden. Deze operaties zijn: interpollering, pronostieken en statistische functies maar ook de *insert*-, *update*- en *delete*-operaties want deze opdrachten kunnen niet meer met de standaard SQL-opdrachten worden uitgevoerd vermits de structuur van de tijdreeksen niet voldoet aan de relationele tabelvorm.

Ter illustratie worden een paar SQL-bevragingen op Tabel 4 getoond, waarbij de tabel is ingevuld met de koersgegevens uit Tabel 1.

Bevraging 1: *Geef alle gegevens van het aandeel met sleutel 1.*

```
SELECT * FROM tabel4 WHERE Sleutel=1;

1 SCOB UA (2 jan 1999 - 456, NULL, 4 jan 1999 - 462, 5 jan 1999 - 464)
```

Het resultaat van de bevraging bestaat uit één rij, waarvan de eerste drie kolommen (*Sleutel*, *Naam* en *Emittent*) overeenkomen met het resultaat van een analoge bevraging in SQL zonder het *Tijdreeks*-type. De vierde kolom is echter van het type *Tijdreeks*[2]. De *NULL*-waarde duidt op het ontbreken van een koers op 3 januari 1999.

Bevraging 2: *Interpolleer de koersgegevens van het aandeel met sleutel 1.*

```
SELECT Interp(Koers) FROM tabel4 WHERE Sleutel=1;

(2 jan 1999 - 456, 3 jan 1999 - 459, 4 jan 1999 - 462, 5 jan 1999 - 464)
```

Het resultaat is géén tabel met koersen maar één *Tijdreeks*-instantie waarin de *NULL*-waarde op 3 januari 1999 vervangen is door *3 jan 99 - 459* (459 is de lineaire interpollatie van 456 en 462). De meeste databanken voorzien echter wel een operatie die een *Tijdreeks*-instantie omzet naar een relationele tabel. De omgekeerde omzetting van tabel naar een *Tijdreeks*-instantie is niet vanzelfsprekend omdat een eventueel patroon van de te vormen tijdreeks moet worden opgegeven.

De eigenlijke onderliggende implementatie van tijdreeksen wordt hier uitdrukkelijk vermeden omdat een efficiënte implementatie volledig ingebed zit in de implementatie van een DBMS en te veel voorkennis over het DBMS vereist. Een tweede reden is dat de implementatie van tijdreeksen bedrijfsvertrouwelijke informatie is die niet zomaar wordt vrijgegeven. De laatste reden is eerder een vermoeden: vermits tijdreeksen zo performant mogelijk verwerkt moeten worden en de eigenlijke werking ervan volledig is afgeschermd van de gebruiker, behoort de broncode waarschijnlijk tot de categorie *Quick and Dirty*: de code moet niet transparant zijn, zolang ze maar correct en efficiënt functioneert.
Toch is het nuttig om dieper in te gaan op de abstracte structuur van tijdreeksen; in deze paragraaf wordt uitgelegd welke parameters nodig zijn om een tijdreeks te typeren. In de analyse van temporele ER-modellen kan dan worden nagegaan of de beschouwde modellen tijdreeksen kunnen ondersteunen.

Er zijn twee categorieën tijdreeksen:

- een *regelmatige* tijdreeks stockeert data-elementen op regelmatige tijdstippen; dit veronderstelt o.a. dat er een voorspelbaar patroon gekend is waarop nieuwe data-elementen kunnen voorkomen, hierdoor wordt de geldigheidsduur van de data-elementen vooraf bepaald.
- een *onregelmatige* tijdreeks stockeert de data-elementen wanneer ze zich voordoen. Een data-element geldt zolang geen nieuw data-element begint te gelden. Voorbeeld van een onregelmatige tijdreeks is de naam van een aandeel, het tijdstip waarop de naam verandert kan niet vooraf vastgelegd worden.

[2] De voorstelling van een tijdreeks in een resultaat verschilt van databank tot databank.

In beide categorieën wordt a priori uitgesloten dat op gelijk welk tijdstip meerdere data-elementen uit dezelfde tijdreeks kunnen gelden[3]. Ook intervallen waarin geen enkel data-element geldt, worden uitgesloten. Toch kan het voorkomen dat er voor een geldige periode geen data-element gekend is. De redenen hiervoor zijn analoog met die van de *NULL*-waarde voor een relationeel attribuut:

- Het attribuut is niet toepasbaar: een aandeel kan wel al bestaan maar nog niet genoteerd staan op de beurs en bijgevolg ook geen wisselkoersen hebben.
- De mogelijke waarde van het attribuut is verloren gegaan en zal dus nooit gekend zijn.
- Wegens uitzonderlijke omstandigheden kan het attribuut onmogelijk een waarde aannemen.

Daarom worden er *excepties* ingevoerd waarmee de gebruiker expliciet meldr dat voor een bepaald tijdstip geen data-element wordt ingegeven.

## *1.4 Tijdreeksen in de SCOB-databank*

Dat de SCOB-databank nuttig gebruik kan maken van tijdreeksen blijkt al onmiddellijk uit de definitie van tijdreeksen door de firma Informix in [INF98]: «*A timeseries is a timestamped series of data entries, such as minute reports of stockprices and trading volumes.*». Aandeelkoersen (*stockprices*) en verhandelde volumes (*trading volumes*) zijn één van de eerst te modelleren concepten uit de SCOB-miniwereld. Toeval of niet maar de meeste lectuur haalt het voorbeeld van aandeelkoersen aan om het nut en de werking van tijdreeksen uit te leggen.

In [GEMI98] zijn alle tijdsafhankelijke attributen afgesplitst volgens de normalisatiemethode uit paragraaf 1.2. Toch kunnen heel wat tabellen gedenormaliseerd worden naar de oorspronkelijke tabel door gebruik te maken van het type *Tijdreeks*. Ter illustratie neem ik de aandeelmodule uit [GEMI98]:

De tabel *Aandeel* heeft als enige tijdsonafhankelijke attributen[4]:

- ***Sleutel***: triviale betekenis
- *AandeelSoort*: winstaandeel, kapitaalsaandeel, etc.
- *Beurs*: de beurs waarop het aandeel genoteerd wordt

In de koerslijsten heeft een aandeel een unieke naam, die echter kan veranderen in de loop der tijden daarom is een aparte tabel *AandeelNaam* aangemaakt met de attributen:

- *Naam*: eigenlijke naam van het aandeel
- ***Aandeel***: referentie naar een aandeelsleutel
- ***Begin***: begin van de periode waarin de naam geldt
- *Einde*: einde van de periode waarin de naam geldt

De naamgegevens voor een aandeel zijn voorstelbaar door een onregelmatige tijdreeks, dus kan de tabel *Aandeel* worden uitgebreid met het attribuut *naam* van het type *Tijdreeks(String)*. Analoog kunnen in de aandeelmodule uit [GEMI98] vrijwel alle tabellen teruggebracht worden naar de tabel *Aandeel*.

De *Aandeel*-tabel krijgt hierdoor een zeer uitgebreide definitie:

- ***Sleutel***, *Soort, Beurs*: tijdsonafhankelijk
- *Naam*, *Emissieprijs*, *Split, Nominale waarde*: onregelmatige tijdreeks
- *Koers*: regelmatige tijdreeks
- *Codering*: onregelmatig, tijdsafhankelijk maar geen tijdreeks.
- …

De praktijk zal moeten uitwijzen of het Oracle8 DBMS in staat is om efficiënt bewerkingen uit te voeren op een tabel met meerdere *Tijdreeks*-attributen[5]. Het is goed mogelijk dat bewerkingen op dergelijke tabellen te veel *overhead* veroorzaken. Indien zulks het geval is, moet een evenwicht gezocht worden tussen twee tegenstrijdige streefdoelen:

- zo veel mogelijk tijdreeksen bijeenbrengen in één tabel
- zo weinig mogelijk *overhead* veroorzaken

Een mogelijk evenwicht zou bereikt kunnen worden door de grote tabel te splitsen in meerdere tabellen waarin bij elkaar horende tijdreeksen gegroepeerd staan: tijdreeksen met algemene informatie (*naam*, *soort*, *codering*, *beurs,* …), met noteringsinformatie (*koers*, *niet-notering*, …) en met kapitaalinformatie (*split*, *nominale waarde*, *aantal aandelen*, …)

[3] Dit is een gevolg van implementatie-eenvoud en de veel voorkomende eis tot momentherleidbaarheid, zie paragraaf 2.2.1.
[4] De vetgedrukte attributen duiden op sleutelattributen.
[5] Het was eigenlijk de bedoeling om dit op voorhand te testen en de resultaten op te nemen in deze thesis maar pas sinds half mei kan het SCOB gebruik maken van een gedeeltelijke Oracle-installatie. Te laat om nog een degelijke test te kunnen uitvoeren.

Het aldanniet noodzakelijk zijn van dergelijk evenwicht is ook van belang in paragraaf 3.7 waar een gelaagd ER-diagramma wordt opgesteld van de SCOB-miniwereld: daar moet beslist worden of de splitsende tabellen effectief worden opgenomen in een ER-model. Wanneer een evenwicht vereist is zullen ze effectief worden opgenomen.

## *1.5 Tijdreeksen als temporele relaties*

In paragraaf 1.1 werd gesteld dat een tijdreeks een opeenvolging van reële waarden is. Het ADT *Tijdreeks* kan echter elk databanktype encapsuleren, inclusief referenties naar sleutelattributen van andere tabellen. Het is dus mogelijk om via tijdreeksen een temporele *N:1*-relatie te definiëren.

Ter illustratie een concept[6] uit de SCOB-miniwereld: op een beurs wordt een aandeel steeds ondergebracht onder één categorie, waarin alle aandelen uit eenzelfde bedrijfstak worden opgenomen. «*Banken*» en «*Hoogovens*» zijn twee voorbeeldcategorieën.

De relatie *Notering* tussen de entiteiten *Aandeel* en *Categorie* heeft de kardinaliteit *N:1*. Indien de relatie tijdsonafhankelijk zou zijn, dan kan het relationele schema als volgt gedefinieerd worden:

```
CREATE TABLE Categorie
( CategorieSleutel INTEGER PRIMARY KEY,
  Naam             VARCHAR(25),
  ...              opsomming van alle andere Categorie-attributen
)

CREATE TABLE Aandeel
( AandeelSleutel INTEGER PRIMARY KEY,
  Naam           VARCHAR(25),
  ...            opsomming van alle andere aandeelattributen
  GenoteerdOp    INTEGER REFERENCES Categorie(CategorieSleutel)
                 ON DELETE SET NULL
                 ON UPDATE CASCADE
)
```

**SQL-fragment 1:** niet-temporele tabeldefinities voor aandeelnoteringen.

Het attribuut *GenoteerdOp* refereert naar de primaire sleutel van de tabel *Categorie* en implementeert zo de genoemde relatie. Telkens wanneer de waarde van het *GenoteerdOp*-attribuut wordt aangepast, controleert het DBMS of de nieuwe waarde effectief refereert naar een bestaande sleutel in de tabel *Categorie*. Het DMBS onderhoudt ook de waarde van de *GenoteerdOp*-attributen wanneer er een *Categorie*-tuppel wordt verwijderd of aangepast.

De relatie *Notering* is echter tijdsafhankelijk want het komt voor dat een aandeel naar een andere categorie wordt verplaatst. De eis dat een aandeel altijd op juist één categorie genoteerd moet staan, dient behouden te blijven. Dit kan verwezenlijkt worden door gebruik te maken van het type *Tijdreeks*.

De creatie van de tabel *Aandeel* ziet er dan als volgt uit:

```
CREATE TABLE Aandeel
( AandeelSleutel INTEGER PRIMARY KEY,
  Naam           VARCHAR(25),
  ...            opsomming van alle andere aandeelattributen
  GenoteerdOp    Tijdreeks(INTEGER)
);
```

**SQL-fragment 2:** tijdreeksdefinitie in de tabel *AANDEEL*.

Probleem hierbij is dat de link tussen de tabellen *Aandeel* en *Categorie* verbroken is. Het attribuut *GenoteerdOp* is louter een tijdreeks van getallen die per toeval refereren naar de primaire sleutel van de tabel *Categorie*. Deze referentie is echter niet meer gekend door het DBMS; voor zover mij bekend, is het ook onmogelijk om deze referentie te definiëren. Wat wel mogelijk is, is om een SQL-*trigger* te definiëren die bij elke *update*-opdracht van

[6] Het concept *aandeel-sector* wordt hier vereenvoudigd voorgesteld. In hoofdstuk 5 wordt dit ten volle uitgewerkt.

een *GenoteerdOp*-waarde controleert of de nieuwe waarde refereert naar een bestaande *Categorie*-instantie. Dit is maar de helft van de oplossing, want bij het verwijderen van *Categorie*-tuppels wordt nog niet gecontroleerd of er *GenoteerdOp*-waarden bestaan die verwijzen naar het verwijderde tuppel. Dit kan weer opgelost worden door over de *Categorie*-tabel een *trigger* te definiëren die bij verwijdering van een tuppel alle tijdreeksen controleert die refereren naar de *Categorie*-tabel. De administrator moet manueel aangeven welke tijdreeksen hierbij gecontroleerd moeten worden[7].

Het is meteen duidelijk dat de voorgestelde methode om tijdreeksrelaties consistent te houden verre van elegant is. Tijdreeksen worden gebruikt om het onderhoud aan de databank te vergemakkelijken maar deze methode maakt het onderhoud een stuk ingewikkelder, zeker wanneer de primaire sleutel van een bepaalde tabel door meerdere tijdreeksen gerefereerd wordt.

Wanneer de mogelijkheid bestaat om een relatie te implementeren via tijdreeksen moet met volgende criteria rekening gehouden worden:
- Het aantal tijdreeksen dat naar eenzelfde primaire sleutel refereert moet minimaal gehouden worden, anders wordt de *trigger* die over deze tabel gedefinieerd moet worden te complex en worden transacties op de tabel onnodig vertraagd vermits telkens alle referende tijdreeksen gecontroleerd moeten worden.
- Het aantal modificatie in de gerefereerde tabel moet zo laag mogelijk worden gehouden. Zo wordt de *trigger*, gedefinieerd over de gerefereerde tabel, zo weinig mogelijk uitgevoerd.
- Wanneer voorspeld kan worden dat de lengte van de refererende tijdreeks laag zal zijn dan is het sowieso beter om geen gebruik te maken van een tijdreeks om de relatie te implementeren; in dit geval zal het efficiënter zijn om de methode van afgesplitste tijdreekstabellen te gebruiken.

Rekening houdend met het feit dat een aandeel zelden naar een andere categorie verhuist, zal het derde criterium van doorslaggevend belang zijn om niet te kiezen voor een tijdreeks in implementatie van de relatie *Notering*.

Het voorgestelde alternatief[8] ziet er dan als volgt uit:

```
CREATE TABLE Aandeel
( AandeelSleutel INTEGER NOT NULL,
  Naam           VARCHAR(25),
  ...            opsomming van alle andere aandeelattributen
  BeginDatum     DATE NOT NULL,
  GenoteerdOp    NUMBER REFERENCES Categorie(CategorieSleutel)
                 ON DELETE SET NULL
                 ON UPDATE CASCADE
  PRIMARY KEY(AandeelSleutel, BeginDatum)
);
```

**SQL-fragment 3:** temporele definitie van de tabel *AANDEEL*.

Het DBMS heeft nu weer de volledige controle over het refererende attribuut *GenoteerdOp*. Vermits het aandeel zelden van categorie zal verhuizen, zullen er ook niet veel tuppels voorkomen voor één bepaald aandeel. Opzoeken op welke categorie het aandeel met sleutel 123 op 3/4/1975 genoteerd staat kan dan gebeuren door volgende bevraging:

```
SELECT GenoteerdOp
FROM Aandeem
WHERE AandeelSleutel=123
  AND BeginDatum<='3/4/1975'
ORDER BY BeginDatum
```

Het eerste tuppel uit de resultaatverzameling levert de gewenste *GenoteerdOp*-waarde.

SQL-fragment 5 en de bijbehorende bevraging maken geen gebruik meer van tijdreeksen maar van geldigheidsduurperiodes. Hiermee wordt het domein van de temporele databanken betreden, hetgeen verder wordt uitgewerkt in hoofdstuk 4.
Het concept tijdreeks is nu voldoende grondig besproken om te kunnen oordelen of bepaalde modelleringstechnieken het concept ondersteunen.

[7] In paragraaf 4.6 wordt deze methode veralgemeend voor temporele attributen, ook degene die niet voorstelbaar zijn door een tijdreeks van referende sleutels.
[8] De tabel *Aandeel* wordt nu gedefinieerd als een geldigheidsduur-tabel; in paragraaf 4.2.2 wordt hierop verder ingegaan.

# **Hoofdstuk 2 :** Het Entity-Relationship model

In [GEMI98] wordt een exhaustieve opsomming gegeven van entiteiten en relaties uit de SCOB-miniwereld. Aan dit werkdocument is een UML-diagramma[9] toegevoegd om de databankstructuur weer te geven. Toch is gebleken dat UML niet echt geschikt is om tijdsafhankelijke relaties duidelijk te visualiseren, daarom is er gezocht naar de mogelijkheden van het klassieke *entity-relationship*-model (afgekort ER-model) en de uitbreidingen ervan om duidelijke visualisaties mogelijk te maken.
Er is gekozen voor het ER-model en niet voor een meer functionele modelleertechniek omdat het beoogde SCOB-model eerder datagericht is en ER zich daar goed toe leent, daarenboven is ER een goed gekend en grondig gedocumenteerd model.

## *2.1 Klassieke ER*

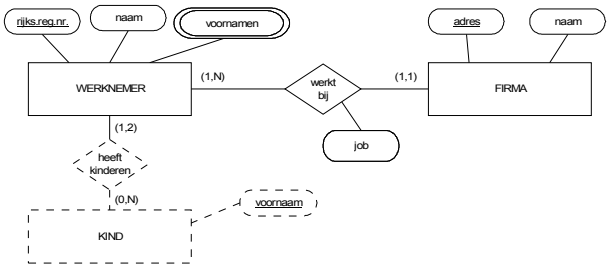
### 2.1.1 Overzicht van de ER-notatie

Hoewel het ER-model een algemeen aanvaarde en veel gebruikte standaard is, zijn er qua notatie toch een paar alternatieven mogelijk. De keuze tussen deze alternatieven is dikwijls afhankelijk van het beschikbare materiaal om ER-diagramma's (afgekort: ERD) op te stellen. Voor dit document is die keuze gebaseerd op de grafische mogelijkheden van het programma *ABC-Flowcharter*[10].

In het verdere verloop van dit document wordt daarom volgende notatie gebruikt voor een klassiek ERD:

- Een *entiteit* wordt voorgesteld door een rechthoek. De naam van de entiteit staat volledig in hoofdletters binnenin die rechthoek.
- Een *relatie* wordt voorgesteld door een ruit. De betrokken entiteiten[11] worden via een lijn verbonden met de ruit. De naam van de relatie staat in kleine letters in deze ruit vermeld. Indien de naam niet eenduidig geïnterpreteerd kan worden, wordt met een pijltje onder de ruit aangeduid in welke richting de naam gelezen moet worden.
- Een *entiteitsattribuut* wordt voorgesteld door een ellips. Deze ellips wordt via een lijn verbonden met de rechthoek van de entiteit. De naam van het attribuut staat in kleine letters binnen de ellips. Bij een sleutelattribuut of een discriminatorattribuut wordt de naam onderlijnd. Meerwaardige attributen worden voorgesteld door een dubbele ellips.
- Een *relatie-attribuut* wordt voorgesteld door een ellips. Het attribuut wordt via een lijn verbonden met de ruit van de relatie. De naam van het attribuut staat in kleine letters binnen de ellips.
- Een zwakke entiteit (*weak entity*) wordt voorgesteld door een rechthoek in stippellijn. Ook de identificerende relatieruit en de verbindingslijnen zijn in stippellijn.
- De *kardinaliteiten* van een relatie worden voorgesteld door bij elke betrokken entiteit een koppel *(min,max)* te vermelden.

Figuur 1 vat deze notatie samen.




**Figuur 1:** samenvattend voorbeeld voor ER-notatie.

[9] UML: *Unified Modeling Language* - voor meer informatie zie [UML97]
[10] ABC-Flowcharter 3.0, Micrografx inc., 1993. Tegenwoordig verkrijgbaar in versie 5.0.
[11] Het is ook mogelijk dat een relatie betrokken wordt op relaties maar dit is niet relevant voor de verdere bespreking van ER-modellen.

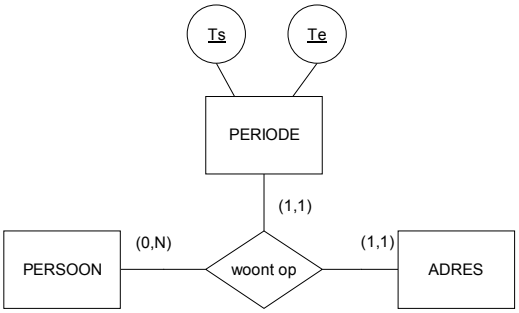
### 2.1.2 Temporele relaties in ER

Een ER-model is bedoeld voor actuele databanken, m.a.w. databanken die enkel de meest actuele toestand van de gemodelleerde miniwereld weergeven. Toch is het ER-model in staat om temporele relaties weer te geven. Neem de *N:1*-relatie *PERSOON woont op ADRES*.


**Figuur 2:** actuele modellering van de relatie *woont op*.

Met deze representatie is het onmogelijk om na te gaan welke adressen een persoon in het verleden heeft bewoond. Toch kan die temporele informatie eenvoudig worden weergegeven in een ERD, als volgt:
- Creëer een entiteit *PERIODE* met sleutelattributen *starttijd* (afgekort: $T_s$) en *eindtijd* ($T_e$).
- Koppel *PERIODE* aan de relatie *woont op*, zodat deze ternair wordt. De entiteit *PERIODE* wordt *temporeelmakend* genoemd.

Nu kan voor elke relevante periode worden weergegeven wanneer een persoon een bepaald adres bewoont (actueel) en heeft bewoond (temporeel).




**Figuur 3:** temporele modellering van de relatie *woont op.*

Meteen duiken er een paar visuele en conceptuele problemen op. Het grootste visuele probleem is de spinnenwebvorm van een ERD met meerdere temporele relaties: de entiteit *PERIODE* zal centraal staan in dergelijk diagramma, omringd door alle temporele relaties en aan de rand van het diagramma alle niet-temporele relaties. De visuele samenhang tussen entiteiten wordt dus voornamelijk bepaald door het wel of niet temporeel zijn, terwijl het in ER juist de bedoeling is om logische clusters te vormen van bij elkaar horende entiteiten.

Dit visuele tekort kan worden verholpen door de entiteit *PERIODE* te ontdubbelen en telkens opnieuw op te nemen. Hierdoor wordt opnieuw een clusterstructuur mogelijk in het ERD, maar dan wordt het model overladen met quasi irrelevant *PERIODE*-entiteiten. De visuele tekorten winnen nog aan invloed wanneer ook temporele attributen worden toegelaten (zie paragraaf 2.1.3). Daarenboven blijft nog het probleem dat binaire temporele relaties in ERD automatisch ternair worden.

Een eerste conceptueel probleem is dat $T_s$ en $T_e$ in het huidige model sleutelattributen zijn, *NULL*-waarden zijn bijgevolg niet toegelaten; hoe moet bijvoorbeeld een periode worden voorgesteld die nog niet is afgesloten of waarvan de begin- of einddatum niet (exact) gekend is?
Een groter conceptueel probleem is dat de (*min*,*max*)-kardinaliteit niet krachtig genoeg is. In het ER-model is aannemelijk dat relatie-instanties *PERIODE*-instanties delen, maar in het relationele model zal dit zelden echt worden toegelaten omdat het temporele gedrag van entiteiten zelden gelijk loopt. Elke instantie van *PERIODE* zal dus maar één keer betrokken worden in een relatie en sowieso de kardinaliteit *1:1* hebben. Neem ter illustratie Figuur 2: een persoon bewoont dus altijd exact één adres en een adres kan geen, één of meerdere personen herbergen. In Figuur 3 is deze semantiek verdwenen want door middel van overlappende periodes kan een persoon meerdere adressen op het zelfde moment bewonen. Persoon *p* woont op adres $a_1$ gedurende de periode *[1990-1998]* en op adres $a_2$ gedurende de periode *[1995-1999]*. De kardinaliteiten zijn voldaan maar toch woont *p* op twee verschillende adressen tijdens de periode *[1995-1998]*.

Hiervoor zijn verschillende oplossingen mogelijk:

- Veronderstel dat een persoon slechts op dagniveau kan verhuizen; hiermee wordt bedoeld dat wanneer een persoon verhuist de vorige *persoon-adres* periode wordt afgesloten op de huidige dag en de nieuwe begint op de volgende dag. In plaats van periodes kan het model dan gebruik maken van *DAG*-entiteiten: aan elke *woont op*-instantie wordt een verzameling *DAG*-entiteiten gekoppeld. Deze verzameling somt alle dagen op waarop een *PERSOON* aan een bepaald *ADRES* gekoppeld is. Het behoeft weinig uitleg om te stellen dat deze oplossing verre van efficiënt is: als een persoon 10 jaar op hetzelfde adres woont dan zullen er meer dan 3650 verschillende *DAG*-entiteiten gekoppeld zijn aan de *woont op* relatie, zelfs het verhogen van een dagniveau naar een weekniveau verhindert een efficïente opslag, bovendien gaat er informatie verloren als een persoon meerdere keren verhuist in één week.
- Het model kan veronderstellen dat periodes elkaar niet kunnen overlappen. Dit is op het eerste gezicht een galante oplossing maar in het geval van de *persoon-adres* relatie moeten de periodes elkaar onmiddellijk opvolgen. Het is niet mogelijk om deze eisen (niet-overlappend of nauw aansluitend) te modelleren in ER zonder de syntax en semantiek van het ER-model uit te breiden.
- Er kan een nieuw type kardinaliteiten ingevoerd worden. Dit gebeurt o.a. in het TER-model, besproken in paragraaf 2.2.5.

In de meeste temporele modellen die verder in dit hoofdstuk besproken worden, ligt het accent van de ER-uitbreiding op temporele relaties.

### 2.1.3 Temporele attributen in ER

Niet enkel relatie-instanties kunnen een temporeel karakter hebben, ook een attribuutwaarde kan met de tijd variëren. De ellipsvoorstelling voor een attribuut in ER wordt beschouwd als een binaire *N:1*-relatie tussen de verzameling entiteitsinstanties en het domein van de attribuutwaarden.
Temporele attributen kunnen dan als volgt worden voorgesteld in ER:

- Reïfieer[12] het attribuut tot een alleenstaande entiteit.
- Verbind de nieuwe entiteit met de oorspronkelijke entiteit door middel van een binaire *N:1*-relatie.
- Koppel de nieuwe relatie aan de entiteit *PERIODE*.

Het visuele gevolg van deze techniek is rampzalig: elke ellips van een temporeel attribuut wordt nu vervangen door twee entiteiten en een ternaire relatie. Een ERD waarin veelvuldig temporele attributen voorkomen, wordt overladen met temporeelmakende relaties die de leesbaarheid van het schema verminderen en die niet echt bijdragen aan de modellering van de miniwereld.

Conceptueel zijn de problemen eenvoudiger, vermits hier enkel de problemen van *N:1*-relaties voorkomen, namelijk dat een temporeel attribuut op gelijk welk tijdstip maar één waarde kan hebben.

## *2.2 Overzicht van temporele ER-uitbreidingen*

### 2.2.1 Inleiding

Sinds de jaren ’80 zijn er heel wat uitbreidingen van het ER-model voorgesteld die temporele modellering mogelijk maken. Deze uitbreidingen kunnen in drie categorieën worden opgedeeld:

1. Er worden visuele afkortingen ingevoerd ter tegemoetkoming aan de visuele problemen van temporeelmakende relaties zoals besproken in 2.1.2 en 2.1.3. Alle temporeelmakende notaties worden hierbij impliciet gemaakt: tijdsattributen en *PERIODE*-entiteiten die louter temporeelmakend zijn, worden niet getekend maar wel verondersteld.
2. De ER-notatie wordt behouden maar de onderliggende semantiek wordt volledig temporeel geherdefinieerd. Niet-temporele relaties en attributen worden vertaald in een temporele variant. Dit heeft het voordeel dat er geen nieuwe notatie moet worden aangeleerd, maar een belangrijk nadeel is dat klassieke ERD’s enkel nog maar syntactisch correct zijn; hun semantiek wordt nu anders geïnterpreteerd waardoor ze geen weerspiegeling meer zijn van de onderliggende miniwereld. Indien de herdefiniëring extreem temporeel gebeurt, dan wordt niet-temporele modellering zelfs onmogelijk.
3. Het ER-model wordt uitgebreid met een aparte syntax en semantiek voor temporele karakteristieken.

De drie categorieën zijn niet wederzijds exclusief, zo kan een bepaald model zowel bepaalde notaties afkorten als bepaalde semantische eigenschappen aanpassen zodat het model in de categorieën 1 en 2 kan worden ondergebracht.

[12] Het werkwoord reïfiëren is afkomstig uit de situatielogica en duidt op het opwaarderen van een situatiepredikaat naar een situatie-object.

Vooraleer verder in te gaan op de verschillende temporele ER-modellen, worden eerst een paar termen gedefinieerd die inherent zijn aan temporele modellering.

**Levensduur** – Temporele entiteiten *leven* gedurende een bepaalde periode in de miniwereld. Het moet mogelijk zijn om deze levensduur weer te geven in het model. Een entiteit kan in leven zijn terwijl bepaalde attributen geen geldige waarde hebben, tenzij een expliciete beperking het anders bepaalt.
Het is aan de ontwerper om vast te leggen of de levensduur één periode is of een reeks periodes. In het laatste geval kan een entiteitsinstantie tijdens bepaalde periodes wel gelden en tijdens andere periodes niet, een instantie kan dus herrijzen.

**Geldigheidsduur** – De geldigheidsduur van een attribuut komt overeen met de periode waarin een attribuutwaarde geldt voor een entiteitsinstantie. De geldigheidsduur is een fundamenteel element voor temporeel gedrag omdat hiermee wordt weergegeven wanneer een entiteitsinstantie bepaalde eigenschappen heeft. Wanneer deze periode niet wordt opgenomen in een model, dan is het model nauwelijks temporeel te noemen.
*Algemene temporele beperking*: indien het levensduur-tijdstype wordt ondersteund dan moet de geldigheidsduur van alle attribuutwaarden van elke entiteitsinstantie een deelverzameling zijn van de levensduur van de entiteitsinstantie.
Wanneer relaties temporeel gemaakt worden, wordt er ook een geldigheidsduur aan gekoppeld. Aan deze geldigheidsduur wordt een vanzelfsprekende beperking opgelegd: de geldigheidsduur van een relatie is een deelverzameling van de doorsnede van de levensduurperiodes van alle deelnemende entiteitsinstanties.
Bv. *PERSOON* en *ADRES* zijn temporele entiteiten (dus met levensduur), *woont op* is een temporele relatie (met geldigheidsduur). *PERSOON* en *ADRES* kunnen maar gerelateerd zijn tijdens de periode waarin beide bestaan. Een adres kan namelijk geen onbestaande personen herbergen en een persoon kan niet wonen op een onbestaand adres.

**Transactietijd** – Dit tijdstype stelt voor wanneer een bepaalde instantie van een attribuut, een entiteit of een relatie wordt opgeslagen in de databank. Het is niet rechtstreeks afhankelijk van de levens- of geldigsheidsduur van gelijk welke instantie in de databank. Transactietijd wordt daarom meestal weggelaten uit een temporeel ER-model vermits de registratie ervan kan worden overgelaten aan het onderliggende DBMS.

Er zijn nog twee bijkomende criteria voor de opbouw van een temporeel ER-model, nl. variabele tijdseenheden en momentherleidbaarheid.

**Variable tijdseenheden** – Het moet mogelijk zijn om de tijdstypes gekoppeld aan de ER-datatypes in verschillende tijdseenheden uit te drukken. Aandeelkoersen moeten per dag kunnen worden opgeslagen dus met de tijdseenheid *dag*, terwijl jaarverslagen slechts jaarlijks gestockeerd moeten worden. Dit is maar een bijkomend criterium omdat de expliciete representatie van tijdstypes en tijdseenheden afhankelijk is van het onderliggende implementatiemodel en bijgevolg minder belangrijk voor de syntax en semantiek van een ER-model.

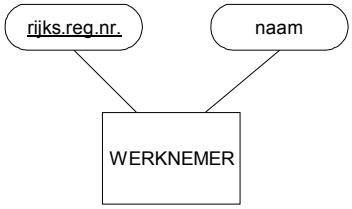
**Momentherleidbaarheid** – Een temporeel ERD is momentherleidbaar wanneer men het voor elk tijdstip *t* kan voorstellen als een niet-temporeel ERD. Deze eigenschap is niet afhankelijk van het gebruikte ER-model maar wel van de gemodelleerde miniwereld; toch kan men in sommige modellen aflezen of een diagramma momentherleidsbaar is of niet. Momentherleidbaarheid is vooral van belang wanneer een temporeel model gebruikt wordt om op bepaald tijdstippen een actuele voorstelling te maken.

Het verdere verloop van dit hoofstuk is voornamelijk gebaseerd op [GREG97] en bevat een bespreking van verschillende ER-uitbreidingen die de afgelopen 10 à 15 jaar zijn ontwikkeld. Hierbij is niet gestreefd naar volledigheid maar wel naar een representatief overzicht waarin ER-modellen uit elk van de drie eerder genoemde categorieën voorkomen.
Voorts is dit overzicht voornamelijk toegespitst op temporele karakteristieken en als bovenlaag van het relationele model zonder temporele eigenschappen noch tijdreeksen. Eventuele andere uitbreidingen zoals een uitgebreider subklasse/superklasse relatietype worden wel vermeld maar niet verder geanalyseerd.

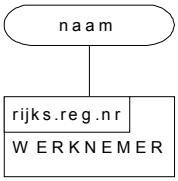
### 2.2.2 RAKE: Relations, Attributes, Keys & Entities

RAKE werd in ’84 voorgesteld als één van de eerste ER-uitbreidingen die temporele modellering moesten mogelijk maken: bepaalde temporele aspecten van een conventioneel ERD worden afgekort of impliciet gemaakt.

Een louter visuele aanpassing van RAKE ten opzichte van klassieke ER is de sleutelbox: sleutelattributen van entiteiten worden niet meer vermeld in een ellips met onderlijnde benaming maar zijn verplaatst naar een sleutelbox in de linker bovenhoek van de entiteitsrechthoek (Figuur 5). Hierdoor worden de sleutelattributen veel explicieter onderscheiden van gewone attributen. Verderop wordt getoond hoe deze aanpassing een temporele voorstelling vergemakkelijkt.



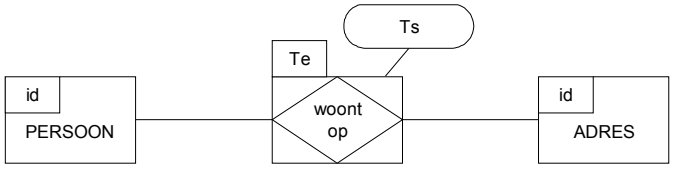

**Figuur 4:** sleutelattribuut in ER.




**Figuur 5:** sleutelattribuut in RAKE.

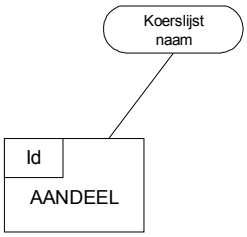
Bij zwakke entiteiten wordt de discriminator in de sleutelbox geplaatst en worden de (primaire) identificerende sleutels in een extra sleutelbox bovenop de entiteitsrechthoek geplaatst.

Een echt temporele uitbreiding in RAKE zijn de attribuutentiteiten (resp. relatie-entiteiten); ze worden voorgesteld door een rechthoek rond de attribuutellips (resp. relatieruit). Attribuut- en relatie-entiteiten duiden op de reïfiëring van attributen (resp. relaties). Om te voorkomen dat het RAKE-model overladen zou worden met (ternaire) temporeelmakende relaties, worden temporele relaties voorgesteld door een zwakke relatie-entiteit die geïdentificeerd wordt door het sleutelattribuut $T_e$. Het attribuut $T_s$ is nu een gewoon attribuut geworden van de binaire relatie-entiteit. Figuur 6 is semantisch equivalent met Figuur 3 op p. 11.

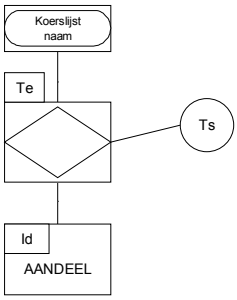



**Figuur 6:** binaire temporele relatie in RAKE.

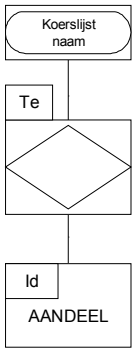
Ook temporele attributen krijgen een nieuwe notatie in RAKE diagramma's. Door het concept *attribuut* te beschouwen als een naamloze *N:1*-relatie tussen entiteit en attribuutwaarde, wordt het attribuut gereïfieerd tot een entiteit; aan deze attribuutrelatie wordt dan opnieuw de entiteit *PERIODE* gekoppeld. Analoog met temporele relaties wordt ook deze attribuutrelatie gereïfieerd.
Voorbeeld: Een aandeel heeft steeds één naam waarmee het op de koerslijst benoemd wordt. Deze naam kan na verloop van tijd wijzigen. De temporele reïfiëring van Figuur 7 wordt getoond in Figuur 8.




**Figuur 7:** niet-temporeel attribuut.




**Figuur 8:** expliciete temporele attribuutnotatie.




**Figuur 9:** impliciete temporele attribuutnotatie.

Voor temporele attributen heeft RAKE nog één extra notationele afkorting, namelijk het relatie-attribuut *Ts* wordt impliciet gemaakt en dus niet vermeld, zie Figuur 9.

RAKE vermijdt dat het volledige ER-model wordt overladen met temporeelmakende ternaire maar voor temporele attributen moet nog steeds een temporeelmakende binaire relatie worden toegevoegd. RAKE lost dus maar ten dele de visuele problemen van temporele karakteristieken in ER op.

### 2.2.3 MOTAR: Model for Objects with Temporal Attributes and Relationships

Motar gaat veel verder dan RAKE wat betreft de uitbreiding van het ER-model. Ten eerste worden de drie bestaande datatypes *entiteit*, *attribuut* en *relatie* geherdefinieerd en ten tweede wordt een nieuw datatype *invariant* ingevoerd. In MOTAR-diagramma's wordt een visueel onderscheid gemaakt tussen temporele en niet-temporele attributen en relaties.

**Entiteitstype** – Er zijn twee soorten entiteiten: enkelvoudige en samengestelde. Beide worden voorgesteld door een cirkel. Samengestelde entiteiten worden opgebouwd uit enkelvoudige en samengestelde entiteiten, door middel van *Component-Geheel* relaties.

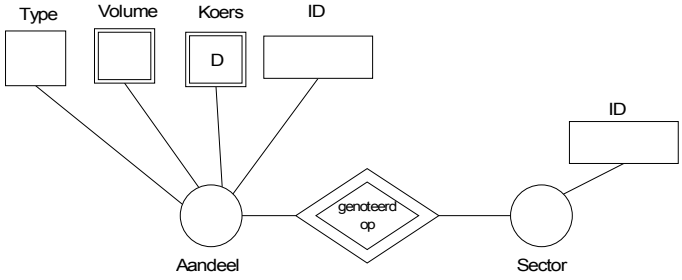
**Attribuuttype** – Dit is onderverdeeld in vier categorieën:
- *Sleutel*: voorgesteld door een rechthoek. Een sleutel in MOTAR is per definitie tijdsinvariant.
- *Periodisch*: voorgesteld door een dubbelzijdig vierkant met een letter die de periode aanduidt: A=*annually* (jaarlijks), M=*monthly* (maandelijks)… Periodische attributen kunnen veranderen van waarde op vooropgestelde tijdstippen. Wanneer de waarde niet verandert, wordt toch een nieuwe (identieke) waarde opgeslagen.
- *Aperiodisch*: voorgesteld door een ongelabeld dubbelzijdig vierkant. Aperiodische attributen veranderen van waarde op onregelmatige tijdstippen. Enkel wanneer de waarde verandert, wordt een nieuwe attribuutwaarde opgeslagen.
- *Algemeen*: voorgesteld door een vierkant. Algemene attributen zijn niet-sleutels die toch tijdsinvariant zijn. Dit is het attribuuttype van traditionele ER.

**Relatietype** – Dit is ook onderverdeeld in vier categorieën:
- *Subklasse-Superklasse*: voorgesteld door een ongelabelde stippellijn tussen twee entiteiten en een pijl gericht van superklasse naar subklasse.
- *Component-Geheel*: voorgesteld door een ongelabelde lijn tussen twee entiteiten en een pijl gericht naar de component.
- *Algemeen*: voorgesteld met de klassieke ER-notatie. Niet-temporele relaties die niet tot de twee bovenstaande behoren, vallen automatisch onder deze categorie. Ook alle relaties met een ariteit groter dan twee zijn van het algemene relatietype.
- *Temporeel algemeen*: voorgesteld met de klassieke ER-notatie behalve dat de relatieruit nu dubbelzijdig wordt getekend.

In MOTAR is het ook mogelijk om kardinaliteiten op te geven voor relatietypes maar de gehanteerde methode is een afzwakking van de *(min,max)*-kardinaliteiten. De kardinaliteiten van de temporele relaties zijn in MOTAR slechts vaag omschreven.

**Invarianttype** – Dit type wordt ook wel *procedurele relatie* genoemd. Ze worden voorgesteld door een holle pijl gericht van conditie naar conclusie. Invarianten worden gebruikt om bepaalde beperkingen op te leggen aan het model.

Ter illustratie wordt volgende miniwereld in MOTAR gemodelleerd: een aandeel heeft een type, een dagelijkse koers en een bepaald verhandelbaar volume en het staat genoteerd op een beurscategorie. Een bedrijf kan beslissen om het verhandelbaar volume te wijzigen.
De modellering is uitgewerkt in Figuur 10. Het attribuut *Volume* is aperiodisch omdat niet van te voren bepaald kan worden wanneer een bedrijf het verhandelbaar volume zal aanpassen. Het attribuut *Koers* is periodisch. De relatie *genoteerd op* is temporeel vermits de categorie-indeling van een beurs af en toe wordt aangepast en een *AANDEEL* dan van *CATEGORIE* verplaatst kan worden.




**Figuur 10:** MOTAR-notatie.

### 2.2.4 TEER: Temporal Extended ER

TEER hanteert integraal de grafische notatie van ER zonder enige visuele uitbreiding. Daarentegen wordt de semantiek van alle datatypes volledig temporeel geherdefinieerd met name door de invoering van de tijdstypes levensduur en geldigheidsduur.

**Entiteitstype** – Aan elke instantie *e* van entiteitstype *E* wordt een temporele karakteristiek *T(e)* gekoppeld die de levensduur van de entiteitsinstantie weergeeft. Deze *T(e)* wordt voorgesteld als één deelinterval of een reeks deelintervallen van het interval *[0, heden]* waarbij 0 een absoluut beginpunt is en *heden* de huidige (steeds voortlopende) tijd voorstelt.
Niet-temporele entiteiten worden gedefinieerd als temporele entiteiten met *T(e)=[0, heden]*

**Attribuuttype** – Het attribuuttype van een entiteit of relatie wordt analoog gedefinieerd. De temporele waarde van elk (temporeel of niet-temporeel) attribuut $A_i$ van een entiteitsinstantie *e* is een partiële functie met als voorschrift $A_i(e):T(e) \rightarrow dom(A_i)$. In TEER wordt deze functie *temporele assignatie* genoemd. De temporele karakteristiek $T(A_i(e))$ van een attribuut is de geldigheidsduur van het attribuut.
Semantiek:

- $A_i(e)$ wordt verondersteld de waarde *NULL* aan te nemen tijdens de intervallen uit $T(e) \setminus T(A_i(e))$, dus voor intervallen waarvoor de temporele assignatie niet bepaald is.
- Een enkelvoudig attribuut *A* van een entiteitsinstantie *e* heeft ten hoogste één waarde op elk tijdstip $t \in T(A_i(e))$. Meervoudige attributen kunnen meerdere waarden aannemen op gelijk welk moment $t \in T(A_i(e))$. De temporele assignatie wordt dan $A_i(e):T(e) \rightarrow 2^{dom(Ai)}$
- De temporele waarde van een samengesteld attribuut is voor elk tijdstip *t* de concatenatie van de temporele waarden van de samenstellende attributen op hetzelfde tijdstip *t*.
- Een sleutelattribuut van entiteitstype *E* is een attribuut van *E* dat op elk moment $t \in [0, heden]$ uniek is voor elke instantie van *E*.

Voorbeeld: De entiteit *PERSOON* heeft attributen *ID*, *voornaam*, *familienaam*, *geboortedatum* en *woonplaats*, waarbij *woonplaats* tijdsafhankelijk is. Een mogelijke *PERSOON*-instantie *p* is dan:

T(p)=[03/04/1975,heden]
ID(p)={[03/04/1975,heden]→7504030224}
voornaam(p)={[03/04/1975,heden]→'Arvid'}
familienaam(p)={[03/04/1975,heden] → 'Claassen'}
geboortedatum(p)= {[03/04/1975,heden]→03/04/1975}
woonplaats(p)={[03/04/1975,12/03/1978]→'Zwijndrecht', [13/03/1978,heden]→'Burcht'}

**Relatietype** – Analoog met de entiteitsinstantie wordt aan elke relatie-instantie een temporele karakteristiek *T(r)* gekoppeld, waarbij *T(r)* een deelinterval is van de doorsnede van de temporele karakteristieken van de betrokken entiteitsinstanties. Er wordt in TEER geen nieuwe betekenis gegeven aan de kardinaliteiten van de betrokken entiteiten.
TEER omvat ook twee soorten superklasse/subklasse-relaties, namelijk *predikaatgedefinieerd* en *manueel gedefinieerd*. In het eerste geval behoort de entiteit *e* van een superklasse *E* ook tot subklasse *F* tijdens alle tijdsintervallen waarvoor het predikaat geldt. In het tweede geval definieert de gebruiker zelf wanneer *e* tot de subklasse behoort. In beide gevallen wordt aan *e* een hiërarchische tijdskarakteristiek *T(e/F)* toegekend waarbij *T(e/F)* noodzakelijk een deelinterval is van *T(e)*.

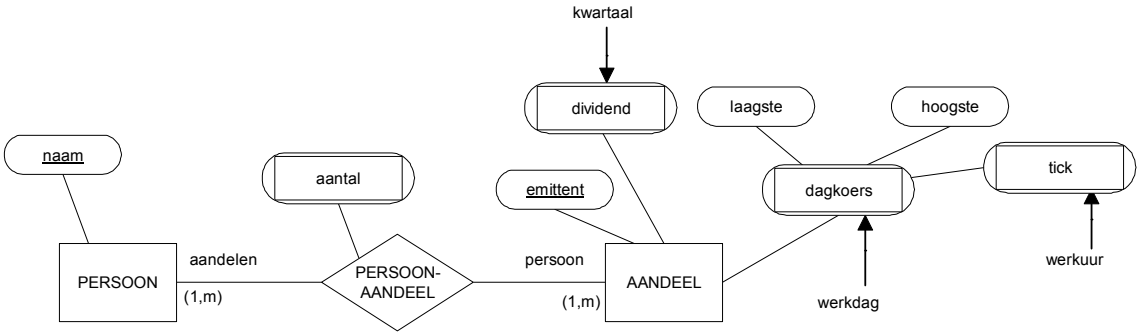
Een recente uitbreiding van het TEER-model is ITDM *(Integrated Temporal Data Model)* waarbij de klemtoon wordt gelegd op het modelleren van tijdreeksen. Om dit mogelijk te maken wordt een nieuwe ER-constructie ingevoerd, nl. het tijdreeksattribuut. Naast een notatie voor ITDM-diagramma's wordt ook een SQL-achtige vraagtaal[13] geïntroduceerd.

In [LEE98] wordt wederom een beursvoorbeeld gebruikt om de ITDM-notatie te demonstreren: de entiteit *AANDEEL* heeft als attributen *emittent*, *dividend* (varieert per kwartaal), *dagkoers* (onderverdeeld in *hoogste* en *laagste,* beide variëren per werkdag) en *tick*[14] (varieert per werkuur). Deze modellering is getekend in Figuur 11.

[13] De vraagtaal van ITDM wordt besproken in paragraaf 4.9.
[14] Een *tick* is een wijziging in de dagkoers.

De attributen *dividend*, *dagkoers* en *tick* zijn gemodelleerd aan de hand van regelmatige tijdreeksen; in een ITDM-diagramma worden deze voorgesteld door een ellips met een inwendige rechthoek. Het periodische patroon dat gekoppeld is aan de regelmatige tijdreeks wordt vermeld door een informele tekst (*kwartaal, werkdag, werkuur*) via een pijl te verbinden met het tijdreeksattribuut. De tijdreeks *dagkoers* is samengesteld uit drie attributen: *laagste, hoogste* en *tick*, waarbij *tick* opnieuw een regelmatige tijdreeks is.

Onregelmatige tijdreeksen worden voorgesteld zoals regelmatige tijdreeksen maar dan zonder vermelding van het patroon.
Relaties worden genoteerd zoals in TEER behalve dat bij de lijn tussen entiteit en relatieruit tevens de rol van de relatie wordt vermeld. Het nut van deze rolvermelding zal duidelijk worden bij de uitwerking van de vraagtaal die ontwikkeld is voor ITDM, zie paragraaf 4.9.




**Figuur 11:** ITDM-diagramma.

### 2.2.5 TER: Temporal ER

TER voorziet het klassieke ER model van nieuwe elementen die temporele modellering mogelijk maken. De klassieke kardinaliteit van een entiteit in relaties wordt daarbij vervangen door twee fundamenteel temporele kardinaliteiten. TER biedt tevens de mogelijkheid om het uiteindelijke diagramma algoritmisch om te zetten naar een relationeel schema. Dit wordt hier echter niet verder uitgewerkt.

**Entiteitstype** – Entiteiten in TER zijn volledig analoog met entiteiten in ER.

**Attribuuttype** – Attributen in TER zijn volledig analoog met attributen in ER. Er zijn dus geen expliciete temporele attributen toegelaten in TER. Dit is echter te omzeilen door een temporeelmakende relatie in te voeren die de entiteit koppelt aan een gereïfieerd temporeel attribuut.

**Relatietype** – Zoals in klassieke ER heeft een relatietype tussen entiteitstypes twee richtingen, telkens van bron naar doel, waarbij de kardinaliteit bij de doelnode wordt geplaatst. In TER wordt de kardinaliteit van een entitiiteitstype ontdubbeld, namelijk in een moment- en een levensduurkardinaliteit.
De momentkardinaliteit, voorgesteld door $M[min_M,max_M]$, bepaalt het minimale en maximale aantal instanties van de doelentiteit dat betrokken mag worden op één instantie van de bronentiteit op elk tijdstip van de bronlevensduur.
De levensduurkardinaliteit, voorgesteld door $L[min_L,max_L]$, bepaalt het minimale en maximale aantal instanties van de doelentiteit dat betrokken mag worden op één instantie van de bronentiteit tijdens de gehele bronlevensduur.

Er is een zeker verband tussen deze twee kardinaliteiten, zoals tot uiting komt in onderstaande ongelijkheden:

- $max_M > 0 \wedge max_L > 0$, anders zou geen enkele instantie van het doelentiteitstype betrokken kunnen worden in de relatie.
- $0 \leq min_M \leq max_M \wedge 0 \leq min_L \leq max_L$, triviaal
- $max_M \leq max_L$, een entiteit kan op een gegeven tijdstip nooit meer keren betrokken zijn in een relatie dan het aantal keer dat de entiteit betrokken is over de gehele levensduur van de entiteit.
- $min_M \leq min_L$, analoog met de max-ongelijkheid

Een instantie van een entiteitstype met $min_S = 0$ is dus niet noodzakelijk altijd betrokken in een instantie van de betreffende relatie, men noemt dit de *optionaliteit* van een entiteitstype. Dankzij het invoeren van de twee temporele kardinaliteiten kan het begrip optionaliteit specifieker behandeld worden.

Een relatietype is *momentoptioneel* a.s.a. $min_S=0$, anders is ze *momentverplicht*. Analoog worden *levensduuroptioneel* en *levensduurverplicht* gedefinieerd.
Het verband tussen deze vier predikaten is:

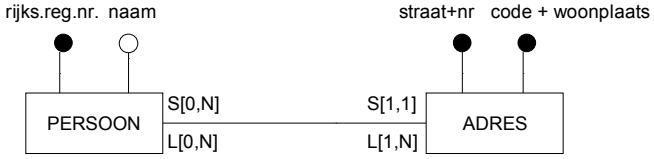
- *levensduuroptioneel* $\Rightarrow$ *momentoptioneel*
- *momentverplicht* $\Rightarrow$ *levensduurverplicht*
- *momentverplicht* en *levensduuroptioneel* kunnen nooit tegelijk gelden

Om de link met niet-temporele kardinaliteiten van de klassieke ER-diagramma's te behouden, wordt in TER de connectiviteit uitgebreid. In klassieke ER is de connectiviteit van een relatie de enkelvoudige kardinaliteit, nl, *1* of *N*, zodat er drie soorten relaties mogelijk zijn, nl. *1:1*, *1:N* of *M:N*. In TER is dit niet meer het geval voor alle mogelijkheden van $min_M$ en $max_M$ in beide relatierichtingen. Voor temporele relaties wordt in TER de $1_t$ connectiviteit toegevoegd die de *één per tijdsinterval* (of ook wel *één per keer*) connectie voorstelt. Hierdoor zijn er nu zes soorten relaties mogelijk, namelijk de drie eerder genoemde plus $1:1_t$, $1_t:1_t$ en $1_t:N$. Toch wordt de connectiviteit zelden gebruikt vermits de levensduur- en momentkardinaliteiten expressiever zijn.

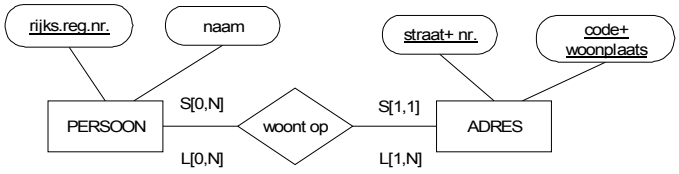
De notatie van TER verschilt nogal van klassieke ER-notatie:

- Entiteiten worden voorgesteld door een gelabelde rechthoek.
- Attributen worden voorgesteld door een kleine cirkel waarnaast de naam staat. Het cirkeltje wordt via een lijn verbonden met een entiteit. Bij sleutelattributen wordt het cirkeltje vol getekend.
- Relaties worden zonder relatienaam voorgesteld door lijnen tussen de betrokken entiteiten.
- De kardinaliteiten worden geplaatst zoals eerder vermeld.

Toch is deze aparte notatie slechts een kwestie van conventie want ze kan zonder verlies van informatie worden omgezet naar de gebruikelijke ER-notatie, waarbij het temporele karakter nog steeds onmiddellijk blijkt uit de dubbele kardinaliteiten. Zo wordt de TER-voorstelling van Figuur 12 volledig analoog met de ER-voorstelling in Figuur 13.




**Figuur 12:** TER-diagramma.




**Figuur 13:** TER-diagramma in ER-notatie.

## *2.3 Beoordeling van temporele ER-modellen*

In [GREG97] worden maar liefst 19 criteria gepresenteerd om een temporeel ER-model te evalueren. In deze paragraaf worden de meest relevante criteria overgenomen en toegepast op de besproken modellen.

1. **Ingebouwde tijdsaspecten:** Levensduur en transactietijd zijn de meest algemene tijdsaspecten van een temporeel model, toch zijn ze geen strikte vereisten. Ten eerste zijn beide aspecten wederzijds onafhankelijk en kan beslist worden om maar één (of zelfs geen) van beide aspecten op te nemen.
   - ER, RAKE, MOTAR: geen ingebouwde tijdsaspecten
   - TEER, ITDM, TER: levensduur

2. **Nieuwe temporele constructies**: Zoals gesteld in paragraaf 2.2.1 zijn er verschillende methodes om het conventionele ER-model uit te breiden, nl. door de semantiek aan te passen, door temporele (semantische) constructies toe te voegen of door een combinatie van de twee. Wat wordt er juist aangepast in de besproken modellen?
   - RAKE, MOTAR: attributen en relaties
   - TEER,ITDM: temporele entiteiten, attributen en relaties
   - TER: moment- en levensduurkardinaliteiten, $1_t$-connectiviteit

3. **Verplicht gebruik van temporele constructies**: Of de gebruiker verplicht is om de temporele constructies van het temporele ER-model te gebruiken (ook voor niet-temporele aspecten) hangt uiteraard mede af van het feit of er veranderingen zijn aangebracht aan de semantiek. In het geval dat de temporele constructies een bovenbouw vormen op het conventionele model, is de gebruiker dus niet verplicht om de temporele constructies te gebruiken. Dit criterium is bijzonder belangrijk voor de gebruiker, zie ook criterium 5.
   - RAKE, MOTAR: optioneel
   - TEER, ITDM, TER: verplicht

4. **Ondersteuning van verschillende tijdsordes**: Verschillende tijdsafhankelijke objecten kunnen worden voorgesteld in verschillende tijdsordes. De koers van een aandeel wordt dagelijks genoteerd maar de kapitaalbalans van een bedrijf wordt slechts jaarlijks genoteerd. Een model dat verschillende tijdsordes ondersteunt kan efficiënter omspringen met tijdsrepresentaties.
   - RAKE, TEER, TER: geen specifieke ondersteuning
   - MOTAR: de gebruiker kan de tijdsorde instellen voor periodische attributen
   - ITDM: voor tijdreeksen kan een temporeel patroon ingesteld worden

5. **Opwaartse compatibiliteit**: Een temporeel model is opwaarts compatibel met het conventionele ER-model wanneer een conventionele modellering zowel syntactisch als semantisch niet verandert in het temporele model. Op zich is dit een zeer streng criterium, ofwel is het schema opwaarts compatibel ofwel niet want zelfs de kleinste semantische aanpassing aan het conventionele model weerlegt de compatibiliteit. Dikwijls zijn de aanpassingen zo minimaal dat men stelt dat een model toch opwaarts compatibel is.
   - RAKE, MOTAR: wel compatibel
   - TEER, ITDM, TER: niet compatibel

[GREG97] onderwerpt alle besproken modellen aan de criteria maar trekt weinig conclusies. De auteurs prefereren modellen die de oorspronkelijke notatie behouden maar de semantiek herdefiniëren. De keuze tussen de verschillende temporele ER-modellen is afhankelijk van de te modelleren miniwereld, met name van volgende twee factoren:
- *aantal* temporele aspecten: Voor een miniwereld waarin slechts weinig aspecten tijdsafhankelijk zijn maakt men het beste gebruik van een model waarin visueel onderscheid wordt gemaakt tussen temporele en niet-temporele constructies.
- *aard* van de tijdsafhankelijkheid: De ontdubbelde kardinaliteiten van het TER-model hebben weinig nut wanneer de tijdsafhankelijkheid volledig door tijdreeksen kan worden voorgesteld.

Welk model nu het meest geschikt is als modelleertechniek, is slechts ten dele objectief te bepalen. Visuele eenvoud is een belangrijke factor maar veel hangt ook af van de modelleerder zelf: in elk model kunnen bijzonder ingewikkelde en onbegrijpbare diagramma's opgesteld worden ook al zijn er binnen hetzelfde model eenvoudige alternatieven.

## *2.4 ACER - Een samenvattend temporeel model*

In deze paragraaf wordt een nieuw temporeel ER-model voorgesteld, namelijk *ACER*[15]; het is geen revolutionair nieuw model maar een combinatie van elementen uit de hiervoor besproken temporele modellen. Een belangrijk element is het temporele attribuuttype tijdreeks uit ITDM. Het ACER-model is voornamelijk gedefinieerd in functie van het relationele model met tijdreeksen. De reden hiervoor is vrij eenvoudig: zoals gesteld in paragraaf 1.4 bevat het SCOB-model heel wat tijdreeksen die veelvoudig gekoppeld worden aan entiteiten.
Om een ER-achtige voorstelling van de SCOB-miniwereld op te stellen, is het aangeraden om een model te gebruiken dat een specifieke representatie hanteert voor entiteiten met o.a. tijdreeksen als attributen maar dat ook minder specifieke temporele modellering mogelijk maakt.

### 2.4.1 Omschrijving van ACER

**Entiteitstype** – Entiteiten in ACER zijn analoog met entiteiten in TEER: aan elke entiteit wordt een levensduur gekoppeld. In TEER wordt de levensduur weergegeven als deelinterval(len) uit *[0, heden]*, in ACER wordt de levensduur bepaald door twee tijdstippen waarbij een tijdstip in de ruimste zin moet worden beschouwd.

[15] Vermits er reeds heel wat ER-varianten zijn ontwikkeld, zijn er ook al heel wat acroniemen in roulatie. Daarom heb ik beslist om mijn initialen voor de letters ER te plaatsen. Er moet dus geen verdere betekenis gezocht worden achter de letter *A* en *C*.

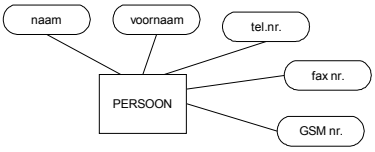
**Relatietype** – Er zijn twee soorten relaties: temporeel en niet-temporeel. De niet-temporele relatie is identiek aan de traditionele ER-relatie. De temporele relatie komt overeen met het relatietype in TER, dus met de ontdubbelde kardinaliteiten.

**Attribuuttype** – Het attribuuttype wordt uit ER overgenomen en uitgebreid met het tijdreekstype. Dit laatste wordt ingevoerd onder twee vormen: het regelmatige en het onregelmatige tijdreeksattribuut. Naast het tijdreeksattribuut wordt ook gebruik gemaakt van minder specifieke temporele attributen, deze kunnen beschouwd worden als een temporele relatie tussen entiteit en attribuutdomein.

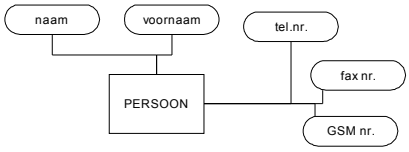
### 2.4.2 Grafische notatie

- Een *entiteit* wordt voorgesteld door een rechthoek. De naam van de entiteit staat volledig in hoofdletters.
- Een *niet-temporeel attribuut* van een entiteit wordt voorgesteld door een ellips. Deze ellips wordt via een lijn verbonden met de rechthoek van de entiteit. De naam van het attribuut staat in kleine letters binnen de ellips. Bij een sleutelattribuut wordt de naam onderlijnd. Meerwaardige niet-temporele attributen worden voorgesteld door een ellips in stippellijn.
- Een *samengesteld niet-temporeel* attribuut wordt voorgesteld door een ellips, verbonden met de rechthoek van de entiteit. Alle samenstellende attributen worden verbonden met de ellips van het samengestelde attribuut.
- Een *onregelmatig tijdreeksattribuut* wordt voorgesteld door een zeshoek, verbonden met de entiteit.
- Een *regelmatig tijdreeksattribuut* wordt voorgesteld door een dubbele zeshoek, verbonden met de entiteit. Boven de zeshoek wordt (desnoods informeel) het temporeel patroon van de tijdreeks genoteerd.
- Een *temporeel niet-tijdreeksattribuut* wordt voorgesteld door een parallellogram, verbonden met de entiteit. Aan de attribuutzijde worden de temporele kardinaliteiten vermeld.
- *Clustering* van attributen: De lijnen van attributen moeten niet rechtstreeks verbonden zijn met de rechthoek van de entiteit, hierdoor wordt clustering van attributen mogelijk. **Figuur 14** en Figuur 15 zijn bijgevolg identiek.
- Een *niet-temporele relatie* wordt voorgesteld door een ruit verbonden met de betrokken entiteiten. In de ruit wordt de naam van de relatie geplaatst. De *(min,max)*-kardinaliteiten worden vermeld zoals bij ER. In recente ER-modellen wordt de relatieruit gewoonlijk niet meer getekend, maar in het SCOB-model komen een paar belangrijke ternaire relaties voor. Zonder ruit als aanduiding wordt het verwarrend om na te gaan welke entiteiten er betrokken zijn in de relatie.
- Een *temporele relatie* wordt voorgesteld door een dubbelzijdige ruit verbonden met de betrokken entiteiten, waarin de naam van de relatie wordt geplaatst. De ontdubbelde kardinaliteiten worden boven de lijn naast de betreffende entiteiten geplaatst als volgt: $[min_M, min_L, max_M, max_L]$

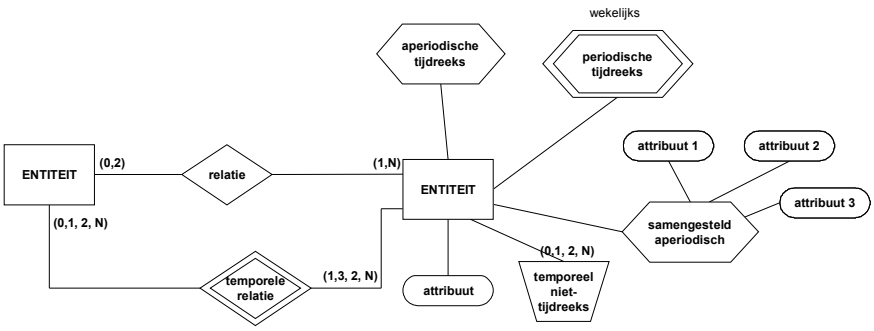
De notatie van ACER wordt samengevat in Figuur 16.




**Figuur 14:** niet-geclusterde attributen.



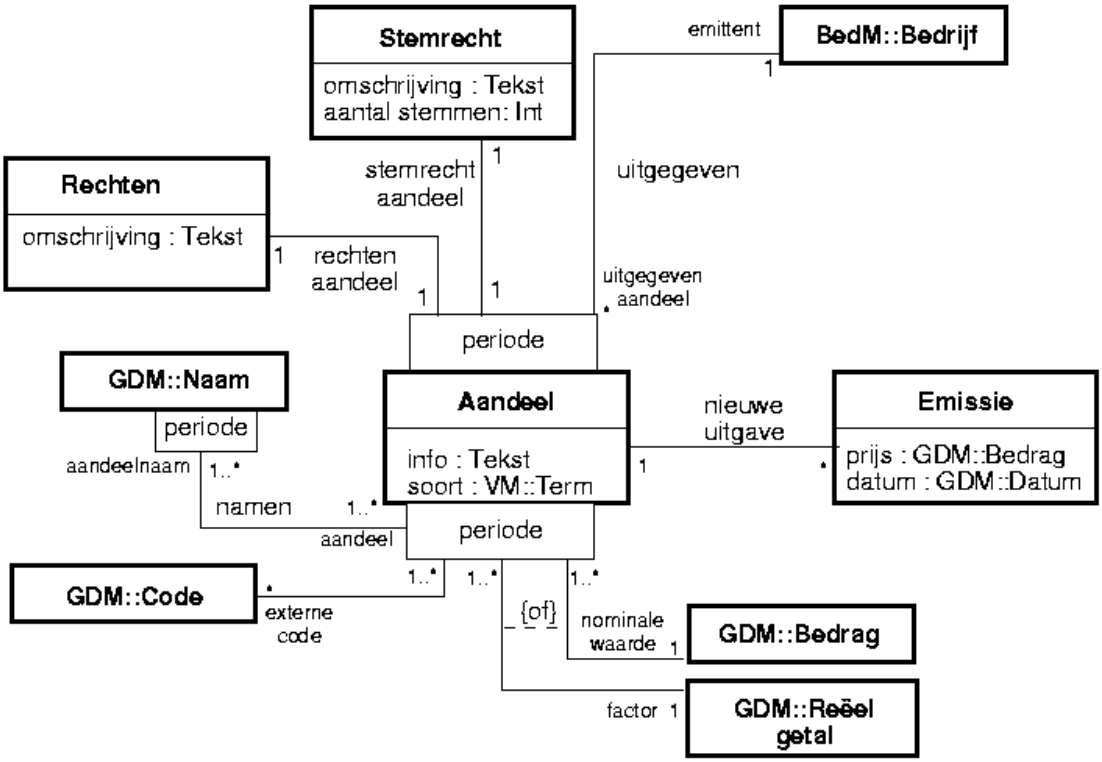

**Figuur 15:** geclusterde attributen.



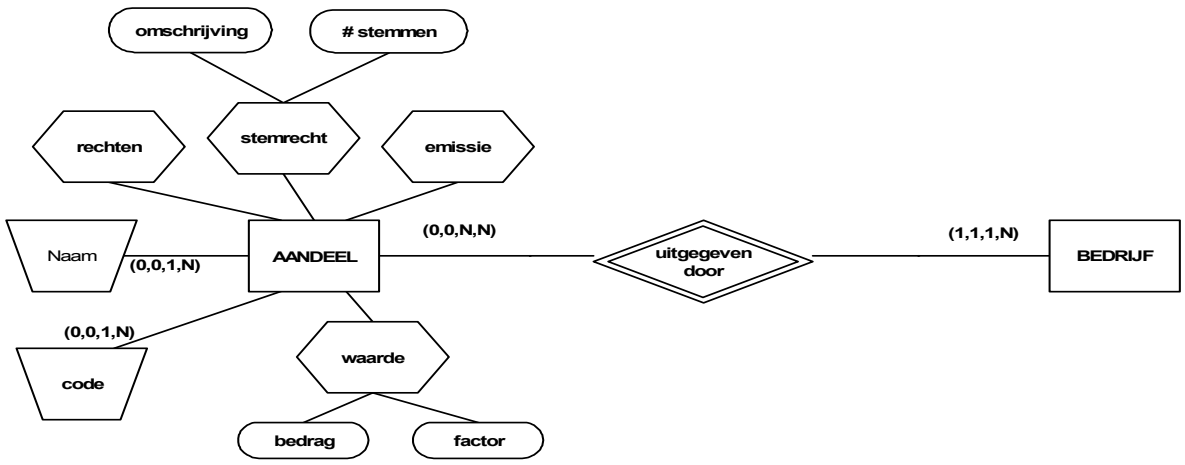

**Figuur 16:** ACER-notatie.

### 2.4.3 Bruikbaarheid

Om de bruikbaarheid van ACER binnen het SCOB-project aan te tonen wordt een UML-diagramma uit [GEMI98] omgezet naar een ACER-diagramma. Het gekozen UML-diagramma (Figuur 17) is één van duidelijkste waarin toch veel informatie is opgenomen. Voor een verdere interpretatie verwijs ik naar [GEMI98] zelf. Figuur 18 is het omgezette ACER-diagramma. Het ACER-diagramma bevat niet helemaal dezelfde informatie, zo geeft de UML een duidelijke typering van de attributen maar zulks is niet eigen aan ER-modellering en is bijgevolg weggevallen in ACER.
Het ACER-diagramma bevat echter meer conceptuele informatie: in Figuur 17 zijn *Stemrecht*, *Rechten*, *Naam*, *Code*, *Bedrag*, *Emissie* en *Bedrijf* aparte entiteiten die gelinkt worden aan de entiteit *Aandeel*, ook al zijn al deze entiteiten, met *Bedrijf* als uitzondering, eigenlijk *Aandeel*-attributen en dus geen entiteiten op zich. *Bedrijf* is samen met *Aandeel* een effectieve entiteit. De link tussen *Aandeel* en *Bedrijf* is dus de enige echte relatie in het schema, maar dat komt niet tot uiting in het UML-diagramma.
Het bevat ook meer temporele informatie: in Figuur 18 is meteen zichtbaar dat de attributen *Emissie*, *Stemrecht*, *Rechten* en *Waarde* aperiodische tijdreeksen zijn, terwijl Figuur 17 niet in staat is om dit weer te geven.




**Figuur 17:** *Aandeel*-modellering in UML.




**Figuur 18:** ACER-modellering van Figuur 17.

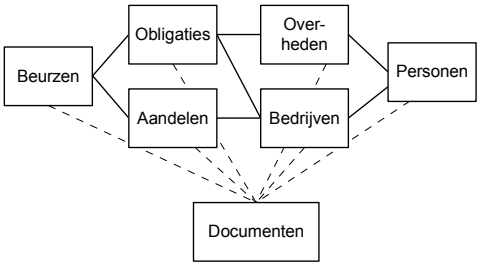
Een troef van ACER is dat alle temporeelmakende relaties impliciet gemaakt zijn. De overblijvende relaties verwijzen bijgevolg naar echte relaties tussen entiteiten uit de miniwereld. Figuur 18 is echter maar een fractie van het volledige ACER-diagramma van de SCOB-databank. In het complete ACER-diagramma komen alsnog te veel relaties voor om een overzichtelijk diagramma te kunnen presenteren. Dit probleem wordt aangepakt in het volgende hoofdstuk.

# **Hoofdstuk 3 :** LAYER - een gelaagd ER-model

Veel temporele modelleerproblemen voor de SCOB-databank zijn door ACER weggewerkt maar de databank is zo uitgebreid dat een volledig en overzichtelijk ACER-diagramma zo goed als onmogelijk is.
In de volgende paragrafen wordt het LAYER-model besproken; LAYER staat voor *Layered ER*, gelaagde ER en is grotendeels gebaseerd op het *Leveled Entity-Relationship-model* (afgekort: LER-model) uit [GAND97]. LAYER is uitgebreider dan LER, vermits LER enkel gedefinieerd is voor binaire relaties terwijl LAYER geen beperking meer oplegt aan de ariteit van relaties.

## *3.1 Inleiding*

De modellering van de meeste miniwerelden kan gebeuren aan de hand van verfijning. De wereld wordt opgedeeld in enigszins onafhankelijke modules die op hun beurt kunnen opgedeeld worden in kleinere groeperingen. Deze methode van verfijning eindigt op het niveau van entiteiten en relaties tussen die entiteiten. In omgekeerde richting kan men spreken van abstractie. Wanneer in dit hoofdstuk wordt gesproken over een niveau, dan wordt de graad van abstractie bedoeld: een hoog niveau betekent dus een hoge graad van abstractie en bijgevolg een lage graad van verfijning.
Ter illustratie neem ik de SCOB-miniwereld: deze wereld kan grosso modo worden opgedeeld in een zevental modules: *Beurzen*, *Obligaties*, *Aandelen*, *Overheden*, *Bedrijven*, *Personen* en *Documenten* (zie Figuur 19). Elk van deze modules wordt dan verder uitgewerkt, zo kan de module *Aandelen* verder worden opgedeeld in een algemeen gedeelte (*naam*, *code*, *soort*), een kapitaalgedeelte (*kapitaalverhoging*, *aandeelruil*, *aantal in roulatie*) en een koersgedeelte (*openingskoers*, *slotkoers*, *hoogste koers* en *laagste koers*). Deze onderverdeling kan dan verder verfijnd worden tot de uiteindelijke entiteiten. Op elk niveau kunnen relaties gelegd worden tussen de abstracte delen onderling of tussen een abstract deel en een abstract deel van een hoger niveau.




**Figuur 19:** hoogste abstractieniveau van de SCOB-databank.

De ER-modellen die tot nog toe beschouwd zijn, bezitten syntax noch semantiek om deze verfijning weer te geven. Een ER-model dat wel zo'n syntax en semantiek omvat, noemt met een gelaagd ER-model omdat er sprake is van verschillende voorstellingen van dezelfde miniwereld, één laag per abstractieniveau waarbij elke lagere laag een meer gedetailleerde versie is van elke hogere laag.

Een gelaagd model biedt een paar voordelen:
- De gebruiker kiest zelf op welk niveau het ER-diagramma wordt weergegeven. De verdere verfijningen blijven verborgen.
- De hiërarchische eigenschappen van het model die niet beschreven worden door een subklasse/ superklasse-relatie kunnen nu wel gevisualiseerd worden.
- De onderliggende logica van het model wordt veel duidelijker.

Bij het opstellen van een gelaagd ER-formalisme duiken wel een paar problemen op:
- Het verband tussen een module op een bepaald abstractieniveau en de verfijnde submodules op een lager niveau moet correct geformaliseerd worden. Een module die wordt verfijnd, noemt men *schil-module* omdat ze als het ware een schil vormt omheen de verfijnde modules. De verfijnde versies van een schilmodule noemt men *kernmodules*. Uiteraard kan een kernmodule opnieuw een schilmodule zijn

bestaande uit verschillende kernmodules van een nog lager niveau. Verderop worden ook de namen *schilentiteit* en *kernentiteit* gebruikt.[16]

- Het verfijnen van een schilmodule kan leiden tot nieuwe relaties op een lager niveau. Deze relaties kunnen betrokken zijn op kernmodules binnen dezelfde schilmodules maar ook op kernmodules van verschillende schilmodules.
- Een relatie tussen een schilentiteit en een andere kernentiteit mag het niveau van abstractie niet breken d.w.z. dat er een mechanisme moet zijn om een op hoog niveau toch toegang te hebben tot details van een (veel) lager niveau zonder dat deze details zijn opgenomen in het hogere niveau.
- De verfijning van een model gebeurt niet automatisch: de ontwerper moet zelf bepalen welke verfijning het best overeenkomt met de logische structuur van de miniwereld.

## *3.2 Informele beschrijving*

Zoals bij het conventionele ER-model, vormen entiteiten de basis van LAYER, attributen worden in LAYER aspecten genoemd en relaties in LAYER komen grosso modo overeen met relaties in ER.

**Entiteiten** – Een entiteit is een object of een concept uit de miniwereld dat een zelfstandig bestaan kent. De verzameling entiteiten kan op verschillende abstractieniveaus worden beschouwd. Ofwel is een entiteit *atomair* conform het ER-formalisme ofwel is er een interne structuur; in het tweede geval wordt de entiteit *abstract* genoemd. Ter verduidelijking een klassiek voorbeeld[17]:

> In de programmeertaal C++ kan op een bepaald abstract niveau een klasse als atomair worden beschouwd, bijvoorbeeld om te controleren of een variabele van een bepaald klassetype is. De interne structuur van de klasse is dan verder van geen belang. In een andere situatie is het belangrijk om de klasse minder abstract te beschouwen, m.a.w. we moeten een niveau dalen in de abstractie. De robuuste C++ klasse wordt verfijnd tot een verzameling methodes. Nog een niveau lager wordt elke methode verfijnd tot een lijst instructies.
> In de andere richting is een klasse een onderdeel van een C++ programma dus zal op een hoger abstractieniveau de klasse als dusdaning niet meer beschouwd worden als alleenstaande entiteit.

Voor elke entiteit wordt een externe structuur opgegeven, dit is de structuur van de entiteit in het niveau waarin de entiteit voor het eerst optreedt. Voor atomaire entiteiten komt de externe structuur overeen met de attributen uit het ER-model. Voor abstracte entiteiten moet het verband tussen de externe en de interne structuur worden weergegeven, dit gebeurt aan de hand van aspecten.

**Aspecten** – Een aspect van een entiteit is de externe weergave van een interne eigenschap. Een aspect maakt dus een deel van de interne concepten van een entiteit zichtbaar voor externe entiteiten, zonder weer te geven hoe het concept intern geconstrueerd zit in de entiteit. Een aspect kan enkel- of meerwaardig zijn, afhankelijk van de onderliggende eigenschap.

Er zijn twee soort aspecten:

- *direct*: Deze weerspiegelen eigenschappen van de entiteit zelf d.w.z. verdere verfijning van de entiteit leidt niet meer tot een nieuwe structuur van het aspect. Directe aspecten zijn te vergelijken met ER-attributen.
- *verdoken*: Deze verbergen nog wel de interne structuur; er is dus nog verdere verfijning nodig vooraleer de eigenlijke aspectstructuur kan worden achterhaald. Verdoken aspecten maken het mogelijk om entiteiten te relateren zonder dat de verdere interne structuur wordt weergegeven. Dit is een belangrijke stap in de ontwikkeling van elk gelaagd ER-model.

Er is nog een andere onderverdeling van aspecten, namelijk volgens domein:

- *waarde-aspect*: De meeste aspecten vallen onder deze categorie. Waarde-aspecten hebben betrekking op een eigenschap uit de gemodelleerde miniwereld. Het domein is een waardetype zoals dat voorkomt in de miniwereld.
- *entiteit-identificerend aspect* (afgekort *eid*-aspect): Om LAYER formeel te kunnen definiëren worden er aspecten ingevoerd die het mogelijk maken om een entiteit te identificeren. Het domein van *eid*-aspecten is modelafhankelijk en verder onbelangrijk voor de bespreking van LAYER[18].

[16] In de literatuur worden ook de termen *subentiteit* en *superentiteit* gebruikt maar om verwarring te voorkomen met het overervings-mechanisme van EER, neem ik deze terminologie niet over.
[17] Een voorbeeld uit de SCOB-miniwereld wordt hier bewust vermeden omdat de verfijning te veel inzicht vergt in de beursterminologie.
[18] In de literatuur wordt het *eid*-attribuut ook wel gedefinieerd door de zgn. functie *surrogate()*.

**Relaties** – Een relatie is een associatie tussen meerdere entiteiten. In LAYER wordt zo'n associatie weergegeven met een link tussen *eid*-aspecten van de betrokken entiteiten. Dit komt overeen met het relatieformalisme in ER, vermits ook daar de entiteiten worden geïdentificeerd door (impliciete) *eid*-attributen.
Een speciaal geval van relaties is de verfijningsrelatie die een schilentiteit associeert met kernentiteiten.
Een instantie van een relatie betrokken op een meerwaardig aspect wordt beschouwd als een reeks relatie-instanties die afzonderlijk betrokken zijn op een andere waarde van het meerwaardige aspect.

## *3.3 LAYER: Formele definitie*

Volgende verzamelingen worden als gekend verondersteld:

$N_a$: verzameling aspectnamen
$N_i$: verzameling van *eid*-aspectnamen waarbij $N_i \subset N_a$
$N_e$: verzameling entiteitnamen waarbij $N_e \subset N_a$
$N_r$: verzameling relatienamen
*EID*: verzameling van alle entiteit-identificaties
*W*: verzameling van alle mogelijke aspectwaarden

Voor de duidelijkheid: $N_a$, $N_i$, $N_e$ en $N_r$ zijn naamverzamelingen, *EID* en *W* zijn waardeverzamelingen.
De verzamelingen $N_a$, $N_r$, *EID* en *W* zijn aftelbaar oneindig en paarsgewijs disjunct.
De elementen uit *EID* en *W* worden geschikt in de domeinverzameling $D=\{EID, D_1, D_2,\ldots\}$ met $\forall i \bullet D_i \subset W$ waarbij de verzamelingen $D_i$ niet noodzakelijk paarsgewijs disjunct zijn; de domeinen mogen dus overlappen.

Een LAYER-schema wordt gedefinieerd met behulp van aspectnamen, relatienamen en domeinen tezamen met functies die het onderlinge verband vastleggen.

Formeel is een LAYER-schema een 6-tuppel $(E,A,R,\alpha,\rho,\varepsilon)$

- *E*: eindige verzameling entiteitsnamen, $E \subset N_e$
- *A*: eindige verzameling aspectnamen, $A \subset N_a$
- *R*: eindige verzameling relatienamen, $R \subset N_r$
- $\alpha$: een totale functie die elke aspectnaam uit *A* afbeeldt op een koppel *(domein,type)* waarbij *type* de waarde *e* (voor enkelwaardig) of *m* (voor meerwaardig) kan aannemen.
  Formeel - $\alpha: A \rightarrow D_i \times \{e,m\}$
  Eigenschap: $\forall n \in N_i \bullet \alpha(n)=(EID,e)$ d.w.z. dat elk *eid*-aspect steeds enkelwaardig is.
- $\rho$: een totale functie die een relatienaam afbeeldt op een vector *eid*-aspecten.
  Formeel - $\rho: R \rightarrow (n,E^n)$ waarbij *n* het aantal entiteiten is dat betrokken is op de relatie met $n \geq 2$, $E^n$ is de vector met namen van de betrokken entiteiten
  In het oorspronkelijke LER-model werd $\rho$ gedefinieerd als: $\rho: R \rightarrow E \times E$, wat duidelijk enkel binaire relaties toestaat.
- $\varepsilon$: een totale functie die elke entiteit afbeeldt op een (eventueel lege) interne structuur. Het is deze functie die gelaagdheid in LAYER mogelijk maakt.
  Formeel – $\varepsilon: E \rightarrow (E_E, A_E, R_E, V_E)$ *met*
  - $E_E$: de verzameling kernentiteiten binnen een entiteit uit *E*, $E_E \subset E$
    Om latere definities en eigenschappen te vereenvoudigen wordt hier de verzameling $B_{Ei}$ van de aspecten binnen een entiteit $E_i$ uit *E* gedefinieerd:
    $$B_{E_i} = E_i \cup \bigcup_{E' \in E_E} A'_E$$
  - $A_E$: de verzameling aspecten van een entiteit uit *E*, waarbij $A_E \subseteq A$ *en* $E \in A_E$
  - $R_E$: de verzameling relaties binnen een entiteit $E_i$ uit *E*
    $\forall r \in R_E: \rho(r)=(n,(eid\text{-}aspect_1, \ldots, eid\text{-}aspect_n)) \Rightarrow eid\text{-}aspect_i \in B_{Ei}$ voor $1 \leq i \leq n$
  - $V_E$: een partiële functie die de aspecten van entiteiten uit *E* afbeeldt op de verdoken aspecten binnen die entiteiten en zo het verband vastlegt tussen de externe en de interne structuur van een entiteit. Dit verband wordt een *correspondentie* genoemd.
    Formeel – $V_E: A_E \setminus \{E\} \rightarrow B$
    $V_E$ is enkel gedefinieerd voor verdoken aspecten.

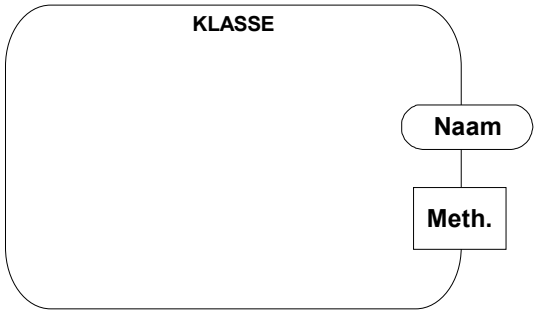
  Voor een atomaire entiteit geldt $E_E=R_E=V_E=\varnothing$. *De verzameling* $A_i$ kan onmogelijk leeg zijn vermits elke entiteit op zijn minst één *eid*-aspect moet hebben

Ter illustratie volgt nu het LAYER-schema van het C++ klassevoorbeeld uit de vorige paragraaf:

- *E={METHODE, DEFINITIE, KLASSE}*
- *A={Methode, Naam, Blok, Declaratie}* $\cup$ *E*
- *R={gedefinieerd_als, heeft_methode}*
- $\varepsilon$*(METHODE)={{METHODE,Naam},*$\varnothing,\varnothing,\varnothing$*}*
  $\varepsilon$*(DEFINITIE)={{DEFINITIE,Blok},*$\varnothing,\varnothing,\varnothing$*}*
  $\varepsilon$*(KLASSE)={{KLASSE,Naam,Methode},{METHODE},{heeft_methode},{(2,(Methode,METHODE))}}*
- $\alpha$*(METHODE)=(EID,s)*
  $\alpha$*(DEFINITIE)=(EID,s)*
  $\alpha$*(KLASSE)=(EID,s)*
  $\alpha$*(Methode)=(EID,m)*
  $\alpha$*(Naam)=(String, s)*
  $\alpha$*(Blok)=(String, s)*
  $\alpha$*(Declaratie)=(String, s)*
- $\rho$*(gedefinieerd_als)=(2,(KLASSE,DEFINITIE))*
  $\rho$*(heeft_methode)=(2,(KLASSE,METHODE))*

## *3.4 Grafische notatie*

- *Enkelwaardige aspectnamen* worden getekend in een ellips.
- *Meerwaardige aspectnamen* worden voorgesteld door een ellips in stippellijn.
- *Eid-aspectnamen* worden voorgesteld in een rechthoek; elke entiteit heeft een *eid*-aspect maar dit wordt niet meer expliciet vermeld.
- Een *entiteit* wordt voorgesteld in een afgeronde rechthoek met bovenaan de naam van de enteiten. Er zijn echter twee mogelijke representaties van een entiteit, nl. *transparant* of *opaak*. In Figuur 21 wordt de transparante voorstelling gebruikt, d.w.z. dat alle onderliggende structuurdetails van een entiteit worden getoond. Figuur 20 toont de opake voorstelling van de entiteit *KLASSE*, hierbij worden geen interne details meer getekend.
  Bij de transparante voorstelling wordt de naam van de entiteit door een lijn gescheiden van de structuur-weergave.
- *Relaties* worden op de klassieke ER-manier getekend. Toch is er een opmerkelijke aanpassing: een externe entiteit kan niet rechtstreeks worden gerelateerd aan een kernentiteit van een bepaalde schilentiteit, anders zou de gelaagdheid verbroken worden. In Figuur 21 correspondeert het aspect *Meth* van de entiteit *KLASSE* met de entiteit *METHODE*. Een relatie betrokken op de entiteit *METHODE* kan alzo worden gemodelleerd als een relatie betrokken op het aspect *Methode*. *Correspondenties* worden voorgesteld door een vette dubbele pijl tussen de corresponderende elementen.

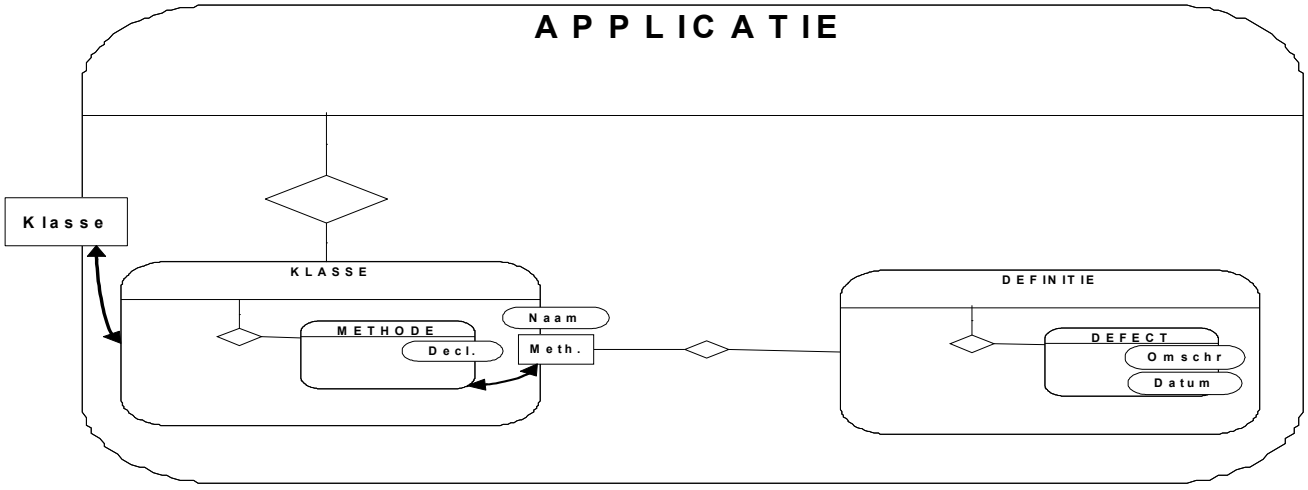



**Figuur 20:** opake voorstelling van de entiteit *KLASSE*.

Ter illustratie wordt het voorbeeld van de C++ klasse uitgebreid.
Deze uitbreidingen zijn:

- de invoering van de entiteit *APPLICATIE* waarbij een applicatie bestaat uit klassen.
- de invoering van de entiteit *DEFECT* waarbij de definitie van een klassse defecten kan bevatten.

Figuur 21 toont een volledig LAYER-diagramma van dit voorbeeld.




**Figuur 21**: LAYER-diagramma van een C++ applicatie.

## *3.5 Een LAYER-instantie*

Een instantie van een LAYER-schema $L(E,A,R,\alpha,\rho,\varepsilon)$ is een koppel functies $(\iota_E,\iota_R)$:

- $\iota_E$ beeldt elke entiteitsnaam uit $E$ af op een eindige verzameling entiteitsinstanties.
  Neem een entiteitsnaam $e$ met $\varepsilon(e)= (E_e, A_e, R_e, V_e)$ en $A_e=\{A_1,\ldots, A_n\}$ dan is een instantie $e_i$ van $e$ een n-tuppel waardeverzamelingen $(W_1,\ldots,W_n)$ met $W_i \subseteq dom(A_i)$, indien $A_i$ een enkelvoudig aspect is, dan is $W_i$ een singelton.
  $W_1$ is de entiteit-identificatie van $e$ of nog $A_1$ is het *eid*-aspect van $e$. Informeel kan $W_1$ gezien worden als een referentie naar zichzelf. Om dit formeel te beschrijven moeten aan $W_1$ beperkingen opgelegd worden. Hiervoor wordt de hulpfunctie $\iota_{eid}$ als volgt gedefinieerd: $\iota_{eid}(e)=\{eid|(eid,\ldots)\in \iota_E(e)\}$

  **Eerste beperking op $W_1$**: $|\iota_E(e)| = |\iota_{eid}(e)|$, wat wil zeggen dat alle instanties van een entiteit $E$ een unieke *eid*-waarde moeten hebben. Binnen een LAYER-instantie is dit niet voldoende, daarom wordt nog een tweede beperking opgelegd.

  **Tweede beperking op $W_1$**: $\iota_{eid}(e)\cap \iota_{eid}(e')=\varnothing$ voor $e\neq e'$. Een *eid*-waarde kan dus niet opnieuw voorkomen in een instantie van een ander entiteitstype.

  Deze twee beperkingen samen stellen dat gelijk welke instantie van gelijk welk entiteitstype een unieke *eid*-waarde heeft. Hierdoor wordt het mogelijk om een eenvoudige evaluatiefunctie $\omega$ te definiëren:
  $\omega(eid,A_i)=W_i$ als $(eid, \ldots, W_i,\ldots)\in\iota_E(e)$, anders is $\omega(eid,A_i)$ niet gedefinieerd.

- $\iota_R$ beeldt elke relatienaam in $R$ af op een verzameling relatie-instanties.
  Neem relatienaam $r$ met $\rho(r)=(n,(E_1,\ldots,E_n))$ dan is een relatie-instantie $r_1$ van $r$ een koppel van de vorm $(n,(e_1,\ldots,e_n))$ met $e_i\in dom(\iota_E \nabla E_i)$ hetgeen garandeert dat elke relatie-instantie enkel *eid*-waarden refereert van bestaande entiteitsinstanties en van de correcte entititeitstypes.

Om een instantie te geven van het LAYER-schema voor de C++ klasse, is eerst meer informatie vereist over welke klassen en methodes (inclusief hun definities) voorkomen in de miniwereld.

- Er zijn twee klassen: *Stack* en *Dummy*.
- *Stack* heeft twee methodes: *void Push(Element)* en *Element Pop(void)*, beide hebben de gebruikelijke definities.
- *Dummy* heeft geen methodes.

Dit resulteert in volgende instantie van het LAYER-schema uit de vorige paragraaf:

- $\iota_E$*(METHODE)={ (#3, { void Push(Element)}) , (#4, { Element Pop(void) })}*
  $\iota_E$*(DEFINITIE)={ (#5, "… return;") , (#6, "… return item;")}*
  $\iota_E$*(KLASSE)={(#1,"Stack",{#3,#4}) , (#2,"Dummy",∅)}*
- $\iota_R$*(heeft_methode)={(2,(#1,#3)), (2,(#1,#4))}*
  $\iota_R$*(gedefinieerd_als)={(2,(#3,#5)),(2,(#4,#6))}*

## *3.6 Combinatie van ACER en LAYER tot LACER*

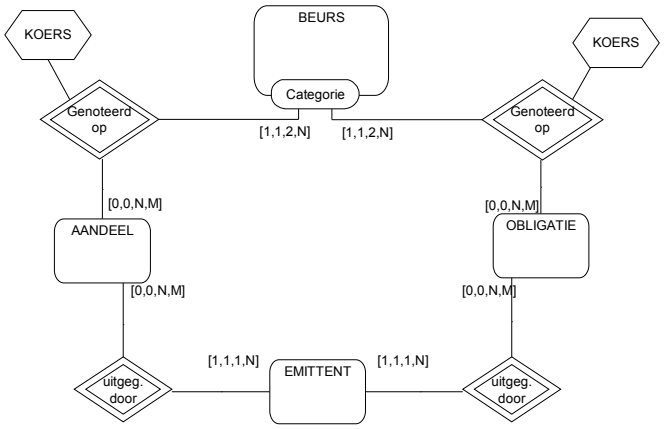
Het ACER-model uit paragraaf 2.4 biedt geen mogelijkheden om een model gelaagd weer te geven. LAYER omvat geen temporele karakteristieken noch kardinaliteiten. Toch zou het SCOB-project handig gebruik kunnen maken van een temporeel, gelaagd ER-model met kardinaliteiten. Daarom wordt in deze paragraaf het LACER-model informeel geïntroduceerd als combinatie van het ACER-model en het LAYER-model; LACER staat voor *layered* ACER, gelaagde ACER.

**Entiteitstype –** Het LAYER-entiteitstype wordt uitgebreid met een levensduur, zoals in TER, ITDM of ACER.

**Attribuuttype** – De LAYER-aspecttypes worden uitgebreid met de verschillende ACER-attribuuttypes, namelijk de regelmatige tijdreeks, de onregelmatige tijdreeks en het temporeel niet-tijdreeksattribuut. Het is ook niet meer verplicht om op een hoger abstractieniveau aspecten te vermelden die enkel van belang zijn op een lager abstractieniveau.

**Relatietype** – De correspondentie-relatie uit LAYER blijft niet temporeel. Daarnaast wordt een temporeel relatietype ingevoerd tussen entiteiten; voor dit type worden de ontdubbelde temporele kardinaliteiten uit ACER gebruikt.

Met deze LACER-omschrijving wordt een eerste abstract model mogelijk van de SCOB-miniwereld:




**Figuur 22:** hoogste abstractieniveau van de SCOB-miniwereld in LACER.

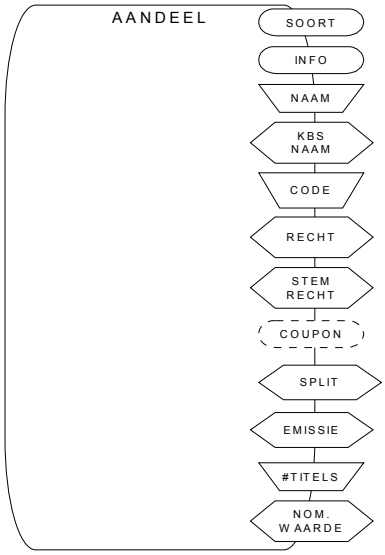
In Figuur 22 worden vier entiteiten opaak getoond: *BEURS*, *EMITTENT, OVERHEID, AANDEEL,*en *OBLIGATIE* zonder verder enig detail te vermelden; behalve dan voor *BEURS*, daar wordt het aspect *Categorie* reeds vermeld omdat zowel *AANDEEL* als *OBLIGATIE* betrokken zijn in een temporele relatie met een *Categorie*. De interne structuur van de entiteit *CATEGORIE* zal terug te vinden zijn op een lager niveau in het LACER-diagramma.

Appendix A toont een uitgewerkt LACER-diagramma van de SCOB-miniwereld tot het laagste niveau van abstractie. Het diagramma is opgenomen in een appendix omdat de eigenlijke modellering te veel inzicht vergt in [GEMI98] en daardoor niet volledig kan besproken worden in het kader van deze thesis.

## *3.7 Eigenschappen*

Een eerste belangrijk kenmerk van LACER is in feite inherent aan modelleren: voor elke niet voor de hand liggende miniwereld bestaat geen eenduidig diagramma, laat staan een eenduidig gelaagd diagramma. In ER kan bijvoorbeeld gekozen worden om een ternaire relatie om te zetten naar drie binaire relaties gekoppeld aan een connectieve entiteit. Bij gelaagde ER-modellering moet bij elk niveau gekozen worden welke onderliggende structuren wel en welke nog niet worden opgenomen op het huidige niveau van abstractie; dikwijls is deze keuze redelijk vanzelfsprekend maar bij twijfelgevallen ligt de beslissing volledig bij de modelleerder, hetgeen sowieso leidt tot uiteenlopende LACER-diagramma's voor één en dezelfde miniwereld.

Ten tweede laat LACER de modelleerder toe om naar believen abstracte niveaus toe te voegen. Zoals verklaard in paragraaf 1.4 kan het zijn dat het onderliggende (mogelijk relationele) model eist dat het aantal complexe attributen (zoals tijdreeksen) per entiteit beperkt moet blijven om een zekere performantie te kunnen garanderen. In diezelfde paragraaf is geprobeerd om een evenwicht te zoeken tussen het aantal attributen voor de entiteit *Aandeel* en het aantal afsplitsingen van groepen bij elkaar horende attributen. Het verschil tussen één entiteit met veel attributen en meerdere entiteiten met gegroepeerde attributen is eenvoudig te visualiseren in LACER. In paragraaf 1.4 zijn de attributen die oorspronkelijk tot de entiteit *AANDEEL* behoren, gegroepeerd in een drietal categorieën: basisinformatie, noteringsinformatie en kapitaalinformatie. Om niet te veel in detail te moeten treden wat de noteringsinformatie betreft, is deze categorie weggelaten. Het resultaat is terug te vinden in Figuur 23 en Figuur 24.

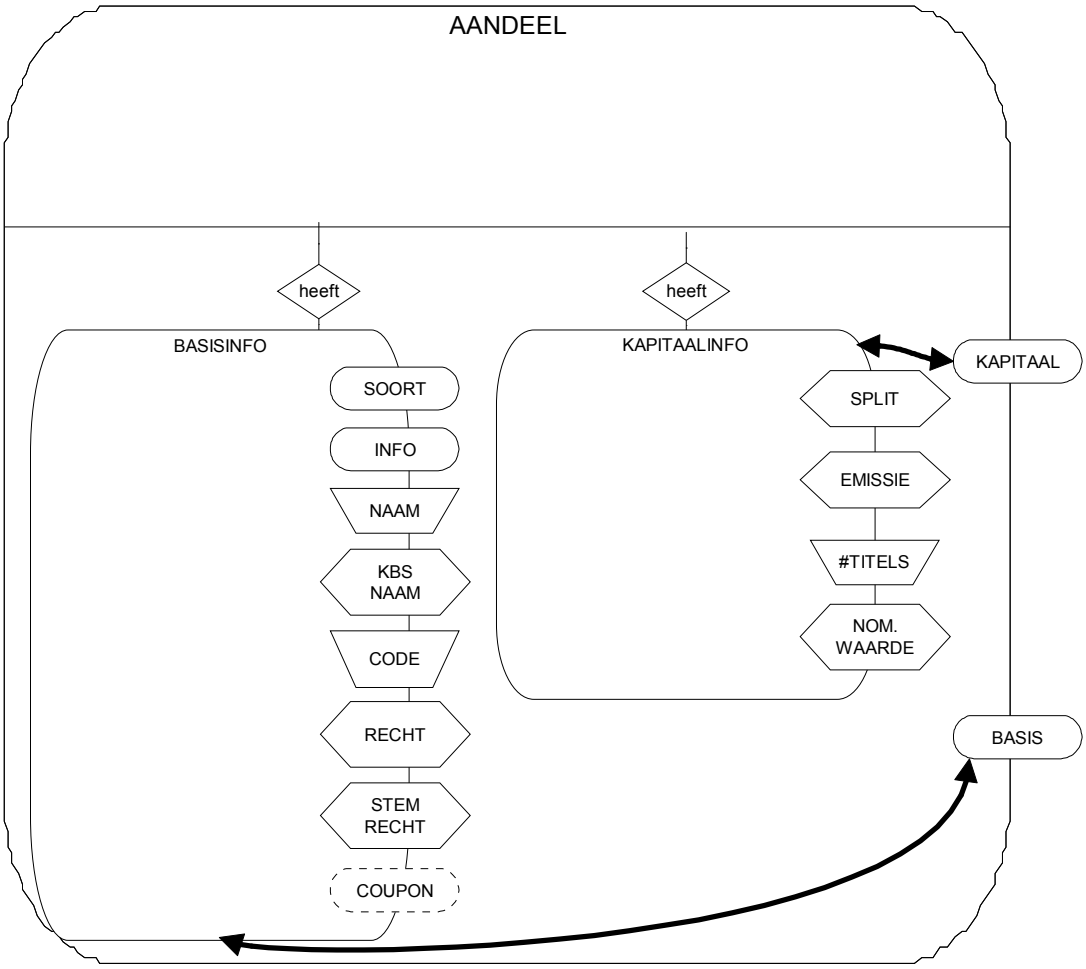



**Figuur 23:** oorspronkelijke entiteit *AANDEEL* met alle attributen.

LACER is geen algemeen model: twee belangrijke concepten uit EER-modellering zijn er niet meer in opgenomen, nl. *IS-A*-relaties en zwakke entiteiten.
Het ontbreken van *IS-A*-relaties is te wijten aan het feit dat LACER reeds te veel gericht is naar een relationele databank waarin de *IS-A*-relaties niet rechtstreeks geïmplementeerd kunnen worden. Zwakke entiteiten kunnen daarentegen niet voorkomen in LACER vermits het onderliggende LAYER-model voor elke entiteit een identificerend *eid*-attribuut heeft.

Uiteraard heeft LACER ook positieve kenmerken: een minder abstracte versie van Figuur 22 hoeft niet noodzakelijk nog alle entiteiten weer te geven. Wanneer het model enkel moet weergeven hoe een aandeel op een beurs genoteerd staat dan volstaat het om uit de figuur de entiteiten *Aandeel* en *Beurs* te lichten en enkel deze entiteiten verder te verfijnen. Men kan dus inzoomen op een bepaalde selectie van entiteiten. Op die manier verkrijgt men naast een gelaagd ER-model ook een modulair model.

Hiermee eindigt het deel over de bespreking van temporele ER-modellen en kan worden overgegaan naar de algemene aspecten van temporele databanken.




**Figuur 24:** entiteit *AANDEEL* met afgesplitste attributen

# Hoofdstuk 4 : Temporele databanken

In dit hoofdstuk worden enkele belangrijke aspecten van temporele databanken besproken; de onderwerpen zijn zo algemeen mogelijk gehouden maar indien mogelijk wordt wel verwezen naar de eventuele toepassingsmogelijkheden op de SCOB-databank.
De paragrafen zijn zeer afwisselend qua inhoud, dit heeft te maken met het feit dat per paragraaf een ander temporeel aspect wordt besproken; er is dus niet altijd een echte samenhang tussen de paragrafen onderling.
Om sommige voorbeelden eenvoudig te houden is gekozen om ze enigszins aangepast over te nemen uit de lectuur (voornamelijk [GREG98], [JENS97], [SNOD97] & [SNOD98]), in plaats van een voorbeeld te puren uit de SCOB-miniwereld.

## *4.1 Inleiding*

De meeste databanken zijn zogenaamde actuele databanken, ook wel *snapshot*-databanken genoemd, waarin enkel de meest actuele situatie van de betrokken miniwereld opgeslagen is. Telkens wanneer de miniwereld veranderingen ondergaat, worden de oude databankwaarden overschreven met nieuwe en kunnen de oude waarden onmogelijk nog achterhaald worden. Het SCOB heeft echter nood aan een historische databank waarbij telkens wanneer de miniwereld veranderingen ondergaat, deze veranderingen zó worden opgeslagen dat het mogelijk blijft om de voorgaande situaties toch te achterhalen; er moet dus een tijdsaspect aanwezig zijn dat aangeeft wanneer een situatie ongeldig wordt of overgaat in een nieuwe situatie. Dergelijke databanken noemt men *temporeel* als tegengesteld aan *actueel*. Een temporele databank wordt algemeen gedefinieerd als een databank die een *notie van tijd* ondersteunt. Met *notie van tijd* wordt niet louter bedoeld dat de databank bijvoorbeeld het SQL-type *TIMESTAMP* ondersteunt want dan zou elke databanktabel die de geboortedatum van personen bijhoudt, temporeel zijn. Er wordt mee bedoeld dat het DBMS in staat is om bijvoorbeeld meerdere versies van een enkelwaardig attribuut te stockeren en afhankelijk van het ingegeven tijdstip te bepalen welke attribuutwaarde gewenst is door de gebruiker.

## *4.2 Soorten temporele databanken*

### *4.2.1* Transactietabellen

Transactietabellen vormen een logboek van een actuele tabel, waarbij elke wijziging (*transactie*) van de opgeslagen gegevens wordt bijgehouden en gedateerd. Aan de hand van dit logsysteem kan de meest actuele toestand van de databank opgevraagd worden; dit is de toestand nadat alle wijzigingen effectief worden uitgevoerd. Maar de gebruiker kan ook de toestand van gelijk welk vroeger tijdstip opvragen, dit is de toestand wanneer enkel die wijzigingen worden uitgevoerd van vóór dat tijdstip.

Een voorbeeld: Op 1 januari 1994 wordt een databank opgestart waarin de hoogte van elke berg uit het Himalaya-gebergte wordt opgeslagen. Eén voor één worden de gegevens ingevoerd en op 4 januari 1994 is *Mount Everest* aan de beurt, hiervoor wordt (foutief) ingegeven dat deze een hoogte heeft van 9888m. Op 9 januari 1994 wordt deze fout opgemerkt en meteen verbeterd naar 8888m. Op 5 mei 1998 blijkt uit een nieuwe meting dat *Mount Everest* slechts 8848m hoog is; deze verbetering wordt op 23 mei 1998 ingegeven in de databank. Een fragment uit de tabel *Berg* ziet er in een transactiedatabank als volgt uit:

| Naam | Hoogte | Transactietijd |
|---|---|---|
| … | … | … |
| Mount Everest | 9888 | 4/1/1994 |
| … | … | … |
| Mount Everest | 8888 | 9/1/1994 |
| … | … | … |
| Mount Everest | 8848 | 23/5/1998 |
| … | … | … |

**Tabel 5:** voorbeeld van de transactietabel *Berg*.

In Tabel 5 is voor elke transactie op de gegevens over *Mount Everest* een rij aanwezig, gedateerd door de kolom *Transactietijd*. Belangrijk om op te merken is dat de datum 5/5/1998 niet is opgenomen in Tabel 5; dit geeft nog eens duidelijk weer dat de inhoud van een transactietabel enkel afhankelijk is van het tijdstip waarop de wijziging wordt doorgevoerd en dus onafhankelijk van het feit dat reeds op 5 mei 1998 een meer correcte hoogte bekend was van *Mount Everest*.
Om bevraging van transactietabellen te vereenvoudigen, wordt meestal nog een attribuut *eindtransactietijd* toegevoegd om aan te duiden wanneer een aanwezige transactie wordt *overschreven* door een nieuwe. Er stelt zich echter een probleem voor transacties die nog niet achterhaald zijn omdat daarvoor nog geen waarde voor het attribuut *eindtransactietijd* kan worden ingegeven. Hiervoor zijn twee veelgebruikte oplossingen: ofwel gebruikt men een waarde *NULL*, ofwel gebruikt men een datum die ver genoeg in de toekomst ligt. Vermits het resultaat van bewerkingen met *NULL*-waarden niet altijd even goed gedefinieerd is, wordt meestal gekozen om de datum 31/12/9999[19] in te vullen als *eindtransactietijd*-waarde. In het laatste geval wordt Tabel 5 uitgebreid met een extra kolom *eindtransactietijd* en als volgt ingevuld:

| Berg | Hoogte | Transactietijd | Eindtransactietijd |
|---|---|---|---|
| … | … | … | … |
| Mount Everest | 9888 | 4/1/1994 | 8/1/1994 |
| … | … | … | … |
| Mount Everest | 8888 | 9/1/1994 | 22/5/1998 |
| … | … | … | … |
| Mount Everest | 8848 | 23/5/1998 | 31/12/9999 |
| … | … | … | … |

**Tabel 6:** transactietabel met dubbele transactietijd.

Zowel in Tabel 5 als in Tabel 6 zijn een paar impliciete veronderstellingen gemaakt:

- Transacties gebeuren blijkbaar enkel op dagniveau; meerdere transacties per dag voor dezelfde berg zijn daardoor niet toegelaten. Indien er op 3/4/1996 drie transacties voor dezelfde berg worden ingegeven dan ontstaan er twee transacties met als transactietijden (3/4/1996, 3/4/1996) waardoor er geen ordening meer mogelijk is tussen de twee. Daarom worden transacties veelal genoteerd in milliseconden of zelfs in microseconden waardoor de kans op transacties met eenzelfde tijdstip verwaarloosbaar wordt.
- De eindtransactietijd van een achterhaalde transactie en de transactietijd van een nieuwe transactie verschillen telkens één tijdseenheid, in dit geval één dag. Dit is een bepaalde keuze want in sommige transactiedatabanken hebben ze een identieke waarde. Het effect van deze keuze wordt duidelijk wanneer we een transactiedatabank bevragen. Dit wordt verderop in deze paragraaf aangetoond.

De gevolgen van deze veronderstellingen zijn vrij beperkt; ze kunnen trouwens per tabel aangepast worden. Het is natuurlijk aangeraden om over de gehele databank dezelfde veronderstellingen te hanteren.

In [SNOD98] wordt een systeem uitgewerkt dat automatische transactielogging mogelijk maakt: wanneer a priori een paar expliciete restricties worden opgelegd aan de mogelijkheden van de actuele tabel dan kan men via SQL een dergelijk systeem implementeren. De gebruiker hoeft dan zelfs niet te weten dat er een transactietabel wordt bijgehouden. Sterker nog, dankzij SQL-*triggers* kan hij zelfs werken op een tabel die geeneens transactie-attributen bevat. Vooraleer hier verder op in te gaan, worden eerst de restricties opgesomd:

- Eens een transactie in de tabel is opgenomen, kan deze niet meer gewijzigd worden. Een foutieve transactie kan bijgevolg enkel gecorrigeerd worden door een nieuwe transactie. In de literatuur wordt een transactietabel ook wel *append-only* genoemd vermits er enkel rijen aan worden toegevoegd.
- Transacties kunnen enkel plaatsgrijpen op systeemtijd. De gebruiker kan een transactie dus niet antidateren. Hierdoor wordt gegarandeerd dat elke nieuwe transactie gebeurt op een tijdstip dat later valt dan gelijk welke transactietijd die zich reeds in de databank bevindt.
- De transactietabel wordt ontdubbeld in een niet-temporele datatabel en een temporele transactietabel. De gebruiker voert enkel transacties uit op de datatabel.

[19] Het gebruik van 31/12/9999 introduceert eigenlijk de Y10K-bug, een variant van de millenniumbug maar dit is hier vrij onbelangrijk omdat tegen het jaar 9999 het datum-type uitgebreid zal zijn en de waarde 31/12/9999 met één *grep*-opdracht opgesppord kan worden.

Het automatische transactiesysteem kan nu als volgt worden opgestart, elke stap wordt toegepast op de *Berg*-databank:

1. Creëer een tabel *Berg* die alle gewenste attributen bevat, exclusief transactie-attributen.
```
CREATE TABLE Berg
( naam VARCHAR(25) PRIMARY KEY,
  hoogte INTEGER,
  …
)
```

2. Creëer een tabel *TT_Berg* die dezelfde attributen bevat, inclusief transactie-attributen. De primaire sleutel wordt nu de primaire sleutel van de tabel *Berg* plus het attribuut *transactietijd*.
```
CREATE TABLE TT_Berg
( naam VARCHAR(25),
  hoogte INTEGER,
  transactietijd  DATE,
  eindtransactietijd DATE,
  PRIMARY KEY (naam, transactietijd)
  …
)
```

3. Creëer een SQL-*trigger* voor elke mogelijke transactie (*insert*, *update* en *delete*) op de oorspronkelijke tabel. In dit voorbeeld onderscheppen deze *triggers* de transacties op de tabel *Berg* en voegen logtransacties toe in de tabel *TT_Berg*. Opvallend (maar wel logisch) is dat een *delete*-opdracht in *Berg* wordt omgezet naar een *update*-opdracht in *TT_Berg* waarbij het attribuut *eindtransactietijd* van 31/12/9999 op een nieuwe waarde wordt gezet, nl. de systeemdatum. Een *update*-opdracht in *Berg* wordt omgezet in twee transacties in *TT_Berg*, nl. een *insert*-opdracht van een nieuwe transactie en een *update*-opdracht van de oude transactie.
   De SQL-*triggers* voor de *Berg*-tabel zien er als volgt uit:

```
CREATE TRIGGER BergInsert
AFTER INSERT ON Berg
REFERENCING NEW AS N
FOR EACH ROW
INSERT INTO TT_Berg(naam, hoogte, transactietijd, eindtransactietijd)
VALUES (N.naam, N.hoogte, SysDate, DATE '9999-12-31')

CREATE TRIGGER BergUpdate
AFTER UPDATE ON Berg
REFERENCING OLD AS O NEW AS N
FOR EACH ROW
BEGIN
   UPDATE TT_Berg
   SET eindtransactietijd=SysDate-1
   WHERE O.naam=TT_Berg.naam
     AND eindtransactietijd='9999-12-31'
   INSERT INTO TT_Berg(naam, hoogte, transactietijd, eindtransactietijd)
   VALUES (N.naam, N.hoogte, SysDate, DATE '9999-12-31')
END

CREATE TRIGGER BergDelete
AFTER DELETE ON Berg
REFERENCING OLD AS O
FOR EACH ROW
UPDATE TT_Berg
SET eindtransactietijd=SysDate-1
WHERE O.naam=TT_Berg.naam
  AND eindtransactietijd='9999-12-31'
```

**SQL-fragment 4:** SQL-triggers voor automatische transactielogging

De gebruiker voert de transacties dus uit op de tabel *Berg* en beoogt daarmee een actuele databank. Het DBMS onderhoudt automatisch de transactietabel *TT_Berg*. Wanneer de gebruiker de gegevens uit de databank wil halen zoals ze zich in de de databank bevonden op een zekere datum (stel 3/4/1994) dan volstaat volgende bevraging:

```
SELECT naam, hoogte
FROM TT_Berg
WHERE transactietijd<=DATE '03/04/1994'
  AND eindtransactietijd>=DATE '03/04/1994'
```

De ongelijkheden in de *WHERE*-clausule van deze bevraging zijn afhankelijk van het feit of gekozen is om het eindtijdstip van de oude transactie en het begintijdstip van een nieuwe transactie gelijk te nemen ofwel van elkaar verschillend met één tijdstip. In het eerste geval moet de tweede ongelijkheid '*strikt groter dan*' worden dus:

```
eindtransactietijd>DATE '3/4/1994'
```

Het nut van transactiedatabanken is drieledig:

- Ten eerste is er het veiligheidsaspect: foutieve transacties kunnen ongedaan gemaakt worden door ze te verwijderen en achterhaalde transacties weer geldig te maken. Het ongedaan maken van transacties heeft tot gevolg dat de inhoud van de actuele tabel *Berg* niet meer overeenkomt met de opeenvolging van de transacties uit de tabel *TT_Berg*. Om deze inconsistentie ongedaan te maken moet de tabel *Berg* opnieuw geconstrueerd worden door één na één alle transacties uit *TT_Berg* uit te voeren op een lege tabel *Berg*.
  Daarenboven kan men naast transactietijden nog meer informatie bijhouden, bijvoorbeeld de naam en de permissies van de invoerder.
- Ten tweede is er het historische aspect: voor sommige vormen van historisch onderzoek is het belangrijk om de inhoud van de databank op een welbepaald tijdstip te kennen. Stel bijvoorbeeld dat er op 3 maart 1997 een meting heeft plaatsgevonden in het Himalaya-gebergte waarbij alle hoogtes relatief bepaald zijn ten opzicht van *Mount Everest*. Het is dan belangrijk om weten wat op 3 maart 1997 de hoogte van *Mount Everest* was in de databank.
- Ten derde zijn er de statistieken: er kunnen grafieken opgesteld worden die bijvoorbeeld per uur weergeven hoeveel transacties er plaatsvinden. Uit dergelijke grafieken kan afgeleid worden hoe het transactieverkeer verloopt doorheen de dag.

Tot nu toe is er nog maar één praktische toepassing gevonden voor een transactiedatabank binnen het SCOB-project: het zou namelijk ideaal zijn indien er een databank opgestart kan worden waaruit men kan opvragen welke beurswetgeving gold op een bepaalde dag sinds 1832. Dit kan gebeuren door eerst alle wetswijzigingen sinds 1832 te verzamelen en deze dan één na één in te geven in een transactiedatabank waarbij de transactietijd geantidateerd wordt naar het tijdstip van de eigenlijke wetswijziging. De eis dat de transacties enkel kunnen plaatsvinden op systeemtijd is hier niet voldaan maar deze eis dient enkel om te garanderen dat elke nieuwe transactie wordt ingevoerd met een transactietijd die na elk reeds aanwezig tijdstip valt; wanneer de wetswijzigingen één na één volgens tijdstip worden ingegeven is, deze eis voldaan. Dit heeft wel tot gevolg dat de invoerder eerst over de totale informatie over wetswijzigingen moet beschikken vooraleer men met de invoer kan beginnen. Dit is in de praktijk een utopie. Vandaar dat een eventuele databank met wetswijzigingen beter kan worden ingevoerd als geldigheidsduur-databank. Over dit type databanken zal de rest van dit hoofdstuk voornamelijk handelen.

### 4.2.2 Geldigsheidsduur-tabellen

Bij geldigheidsduur-tabellen is het niet belangrijk wanneer nieuwe gegevens worden toegevoegd aan de databank maar wel dat zo accuraat mogelijk wordt opgeslagen wanneer de attribuutwaarden gelden.
Zoals besproken in paragraaf 1.2 valt een entiteit die verschillende temporele attributen omvat uiteen in verschillende tabellen: één tabel voor alle niet-temporele attributen en één tabel per temporeel attribuut. Al deze tabellen zijn uitgebreid met twee kolommen *Begin* en *Einde*, die telkens weergeven wanneer een bepaalde waarde geldt.
Een vereenvoudigd voorbeeld uit de SCOB-databank: een *Aandeel* heeft een temporeel attribuut *Naam* en twee niet-temporele attributen *Sleutel* en *Type*. Het kapitaalaandeel met sleutel 1 heeft van 1 januari 1832 t.e.m. 31 december 1843 de naam *Sociéte Générale*, daarna heeft het de naam *Generale Maatschappij* en op 1 april 1853 wijzigt de naam naar *Generale Bank*. Op 1/4/1854 wordt het aandeel van de beurs geschrapt. Aandeel 2 heeft op 1 januari 1832 de naam *Banque Nationale* en die is sindsdien ongewijzigd. Dit geeft aanleiding tot Tabel 7 en Tabel 8.

| AandeelSleutel | Type | Begin | Einde |
|---|---|---|---|
| 1 | Kapitaalaandeel | 01/01/1832 | 31/03/1854 |
| 2 | Aandeel zonder stem | 01/01/1832 | 31/12/9999 |
| … | … | … | … |

**Tabel 7:** niet-temporele attributen van de entiteit *Aandeel*.

| AandeelSleutel | Naam | Begin | Einde |
|---|---|---|---|
| 1 | Sociéte Générale | 01/01/1832 | 31/12/1843 |
| 2 | Banque Nationale | 01/01/1832 | 31/12/9999 |
| 1 | Generale Maatschappij | 01/01/1844 | 31/03/1853 |
| 1 | Generale Bank | 01/04/1853 | 31/03/1854 |
| … | … | … | … |

**Tabel 8:** temporeel attribuut *Naam* van de entiteit *Aandeel*.

Wanneer uit koersboeken blijkt dat aandeel 2 vanaf 4 mei 1866 de naam *Banque de Belgique* krijgt, dan wordt de geldigheidsduur voor de naam *Banque Nationale* afgesloten door het attribuut *Einde* van rij 2 in Tabel 8 de waarde *3 mei 1886* te geven en onderstaande rij toe te voegen:

| | | | |
|---|---|---|---|
| 2 | Banque de Belgique | 04/05/1866 | 31/12/9999 |

Dit is maar één voorbeeld van een transactie op temporele tabellen; deze transacties vergen de nodige aandacht afhankelijk van de temporele attribuutkardinaliteiten; in paragraaf 4.6 wordt dit grondig uitgewerkt. Daar wordt ook meteen verduidelijkt wat het verband is tussen de geldigheidsduur van een aandeel in Tabel 7 en de geldigheidsduur van de *Naam*-attributen van dat aandeel in Tabel 8. In dit geval moeten de geldigheidsduur-intervallen van de *Naam*-attributen samenvallen met de geldigheidsduur van het aandeel en wel zo dat er op elk moment exact één naam geldt voor een aandeel.

### 4.2.3 Bitemporele tabellen

Men kan een geldigheidsduurtabel uitbreiden met een transactielogsysteem. Men spreekt dan van een *bi-temporele* tabel, vermits zowel geldigheidsduur als transactietijden zijn opgenomen.

Ter illustratie neem ik de *Naam*-tabel uit de vorige paragraaf (Tabel 8): stel dat een SCOB-medewerker begint met het ingeven van aandeelnamen. Hij doet dit door de koersboeken jaar na jaar door te nemen en te kijken welke aandelen er bijkomen of van naam veranderen. Op 3 maart 1999 creëert hij het aandeel met *Sleutel*-waarde 1 en geeft dit de naam *Sociéte Générale*. De invoerder weet dat het aandeel zelf een geldigheidsduur heeft van 1/1/1832 t.e.m. 31/3/1854, dus worden volgende gegevens aan de bitemporele tabel toegevoegd:

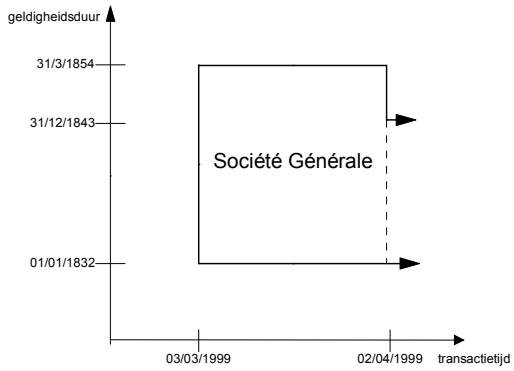
```
Transactietijd: 3/3/1999
Eindtransactietijd: 31/12/9999
Aandeelsleutel: 1
Naam: Sociéte Générale
Begin: 01/01/1832
Einde: 31/03/1854
```

Op 2 april 1999 is de invoerder aanbeland in het koersboek van 1844, waarin hij terugvindt dat de naam van aandeel 1 op 1 januari 1844 wijzigt naar *Generale Maatschappij*. Dus worden analoog met een *update*-opdracht in een transactietabel twee rijen toegevoegd zodat volgende tabel ontstaat:

| AandeelSleutel | Naam | Begin | Einde | Transactietijd | Eind-transactietijd |
|---|---|---|---|---|---|
| 1 | Sociéte Générale | 01/01/1832 | 31/03/1854 | 03/03/1999 | 01/04/1999 |
| 1 | Sociéte Générale | 01/01/1832 | 31/12/1843 | 02/04/1999 | 31/12/9999 |
| 1 | Generale Maatschappij | 01/01/1844 | 31/03/1854 | 02/04/1999 | 31/12/9999 |
| … | … | … | … | … | … |

**Tabel 9:** bitemporele versie van de tabel *AandeelNaam*.

Een bitemporele tabel zoals Tabel 9 is op het eerste gezicht moeilijk te interpreteren maar mits enige oefening kan de lezer de logica ervan doorzien. Een veelgebruikt hulpmiddel is de bitemporele grafiek, waarbij per attribuutwaarde een grafiek wordt getekend met de transactietijd op de X-as en de geldigheidsduur op de Y-as. De twee pijlen naar rechts in Figuur 25 duiden de nu gekende geldigsheidsduur aan. Er zijn meerdere varianten mogelijk op deze grafiekvoorstelling.




**Figuur 25:** bitemporele grafiek voor het *Naam*-attribuut *Société Générale*.

Bitemporele tabellen bevatten dus zowel de geschiedenis van de miniwereld als de invoergeschiedenis van de tabel. Dergelijke tabellen kunnen gebruikt worden bij onderzoek waarbij geweten moet zijn hoe de miniwereld evolueerde en wanneer deze gegevens zijn ingegeven in de databank.

Zoals gezegd is er geen nood aan transactietabellen in de SCOB-databank, dus is er ook geen nood aan bitemporele tabellen. In de rest van dit hoofdstuk ligt de de klemtoon dan ook op geldigheidsduur-tabellen.

## *4.3 Temporele relationele algebra*

Temporele databanken zijn niet zomaar databanken met een speciale aandacht voor tijdsattributen. De semantiek ervan kan onderbouwd worden met een specifieke algebra. Om te kunnen spreken over een temporele relationele algebra dienen de elementaire operaties uit de relationele algebra uitgebreid te worden met temporele aspecten, dit gebeurt in o.a. [BOHL97]. Het betreft hier volgende operaties: *selectie*, *projectie*, *unie*, *carthesisch product* en *verschil*. De *hernoem*-operator komt niet voor in deze opsomming vermits hernoeming onveranderd blijft in de temporele algebra.
Een temporeel tuppel wordt voorgesteld als $<x \,||\, [S \rightarrow E]>$ waarbij *x* alle expliciete attributen voorstelt en $[S \rightarrow E]$ de geldigheidsduur voorstelt.
In deze paragraaf wordt een kleine letter *r* gebruikt voor een relatie-instantie van *R*.

Temporele selectie:

$\sigma_{t,cond}\, r = \{<x \,||\, [S \rightarrow E]> \bullet <x \,||\, [S \rightarrow E]> \in r \wedge cond(<x \,||\, [S \rightarrow E]>)\}$

*cond* is de selectieconditie, van de vorm $A_1\ \Theta\ A_2$ of $A_1\ \Theta\ c$, waarbij $A_1$, $A_2$ attributen zijn, *c* een constante of een aggregatie en $\Theta \in \{=, \neq, <, \leq, >, \geq\}$.
$dom(A_1)$, $dom(A_2)$ en $dom(c)$ zijn met mekaar vergelijkbaar.

Temporele projectie:

$\pi_{t,f}\, r = \{<x_1 \,||\, [S \rightarrow E]> \bullet <x_2 \,||\, [S \rightarrow E]> \in r \wedge x_2 = f(<x_1 \,||\, [S \rightarrow E]>)\}$

De afbeelding *f* bepaalt de attributen die geprojecteerd worden, de waarde van de attributen blijft ongewijzigd.

Temporele unie:

$r_1 \cup_t r_2 = \{<x \,||\, [S \rightarrow E]> \bullet <x \,||\, [S \rightarrow E]> \in r_1 \vee <x \,||\, [S \rightarrow E]> \in r_2\}$

De definities van temporeel carthesisch product en temporeel verschil zijn niet helemaal analoog met de niet-temporele versies vermits de geldigheidsduur van de oorspronkelijke tuppels niet noodzakelijk behouden blijft.

Temporeel carthesisch product:

$r_1 \times_t r_2 = \{ <x_1, x_2 \,||\, [S \rightarrow E]> \bullet <x_1 \,||\, [S_1 \rightarrow E_1]> \in r1 \wedge <x_2 \,||\, [S_2 \rightarrow E_2]> \in r2 \wedge S = max(S_1, S_2) \wedge E = min(E_1, E_2) \wedge S \leq E \}$

Het carthesisch product van twee relaties is elke mogelijke concatenatie van een tuppel uit $R_1$ en een tuppel uit $R_2$ waarbij de geldigheidsduur-intervallen van beide tuppels elkaar overlappen. De intuïtie achter deze definitie is de volgende: twee tuppels kunnen enkel met mekaar geconcateneerd worden tijdens het interval waarbinnen beide tuppels gelden.

Temporeel verschil:

$$r_1 \setminus_t r_2 = \{ <x \,||\, [S \rightarrow E]> \bullet <x \,||\, [S_1 \rightarrow E_1]> \in r_1 \wedge ((\exists S_2 \bullet <x \,||\, [S_2 \rightarrow S]> \in r_2 \wedge S_1 \leq S) \veebar S = S_1) \wedge ((\exists E_2 \bullet <x \,||\, [E \rightarrow E_2]> \in r_2 \wedge E_1 \geq E) \veebar E = E_1) \wedge \neg \exists S_3, E_3 \bullet < x \,|\, |[S_3 \rightarrow E_3]> \in r_2 \wedge E_3 > S \wedge S_3 < E \}$$

Het symbool $\veebar$ staat voor de logische *exclusieve of*. Het temporeel verschil trekt van de geldigheidsduur van elk tuppel $t_1$ uit $r_1$ de geldigheidsduur van alle tuppels uit *r2* met identieke attribuutwaarden af. Hierbij kan het gebeuren dat een tuppel uit *r1* wordt opgesplitst in meerdere tuppels.

Vb: neem de relatie-instanties $r_1 = \{<a \,||\, [2 \rightarrow 9]>\}$ en $r_2 = \{<a \,||\, [5 \rightarrow 7]>\}$, het temporeel verschil is dan:

$r_1 \setminus_t r_2 = \{<a \,||\, [2 \rightarrow 5]>, <a \,||\, [7 \rightarrow 9]>\}$.

Analoog met de niet-temporele relationele algebra kunnen aan de hand van de geherdefinieerde temporele operatoren nieuwe temporele operatoren gedefinieerd worden:

Temporele join:

$r_1 \_ _{t,\, cond}\, r_2 = \sigma_{t,cond}\, (r_1 \times_{t,\, cond}\, r_2)$

Temporele deling:

Voor relatieschema's *R1* $<X, Y \,||\, [S \rightarrow E]>$ en *R2* $<X \,||\, [S \rightarrow E]>$ geldt:

$r1 :_t r2 = \pi_{t,Y}\, r_1 \setminus_t \pi_{t,Y}\, ((r_2 \times_t \pi_{t,Y}\, r_1) \setminus_t r_1)$

## *4.4 Duplicaten*

Het onderhoud van temporele tabellen verloopt niet analoog met het onderhoud van niet-temporele tabellen. Het vermijden van duplicaten is hiervan een voorbeeld. Het document [SNOD98] vormt een basis voor deze paragraaf maar ik probeer het geheel formeler weer te geven.

In niet-temporele tabellen onderscheidt men twee types duplicaten:

- *sleutelduplicaten*: tuppels met identieke sleutelattributen
- *attribuutduplicaten*: tuppels waarvan alle attributen identiek zijn

Bij temporele tabellen onderscheidt men vijf types:

- *sleutelduplicaten*: tuppels met identieke sleutelattributen
- *waardeduplicaten*: tuppels waarvan alle niet-temporele attributen identiek zijn; de geldigheidsduur van beide tuppels wordt buiten beschouwing gelaten.
  Formeel: $<x_1, \ldots, x_n \,||\, [S_x \rightarrow E_x]>$ en $<y_1, \ldots, y_n \,||\, [S_y \rightarrow E_y]>$ zijn waardeduplicaten a.s.a. $x_i = y_i$ voor $1 \leq i \leq n$.
  De eerste drie rijen uit Tabel 10 zijn waardeduplicaten.
- *intervalduplicaten*: waardeduplicaten waarvan de geldigheidsduur een tijdsinterval gemeenschappelijk heeft.
  Formeel: $<x_1, \ldots, x_n \,||\, [S_x \rightarrow E_x]>$ en $<y_1, \ldots, y_n \,||\, [S_y \rightarrow E_y]>$ zijn intervalduplicaten a.s.a. $x_i = y_i$ voor $1 \leq i \leq n \wedge \exists t \bullet (t \in [S_x \rightarrow E_x] \wedge t \in [S_y \rightarrow E_y])$.
  De vierde en vijfde rij uit Tabel 10 zijn intervalduplicaten over de periode van 1 januari 1999 t.e.m. 12 januari 1999.
- *attribuutduplicaten*: tuppels waarvan alle attributen identiek zijn
  Formeel: $<x_1, \ldots, x_n \,||\, [S_x \rightarrow E_x]>$ en $<y_1, \ldots, y_n \,||\, [S_y \rightarrow E_y]>$ zijn attribuutduplicaten a.s.a. $x_i = y_i$ voor $1 \leq i \leq n \wedge S_x = S_y \wedge E_x = E_y$.
  Attribuutduplicaten zijn een speciaal geval van intervalduplicaten.
  De eerste twee rijen uit Tabel 10 zijn attribuutduplicaten.

- *actuele duplicaten*: waardeduplicaten waarvan de geldigheidsduur-periodes de systeemtijd gemeenschappelijk hebben.
  Formeel: $<x_1, \ldots, x_n \,||\, [S_x \rightarrow E_x]>$ en $<y_1, \ldots, y_n \,||\, [S_y \rightarrow E_y]>$ zijn actuele duplicaten a.s.a. $x_i = y_i$ voor $1 \leq i \leq n \wedge$ *systeemtijd* $\in [S_x \rightarrow E_x] \wedge$ *systeemtijd* $\in [S_y \rightarrow E_y]$.
  Actuele duplicaten zijn een speciaal geval van intervalduplicaten met een bijzondere eigenschap: actuele duplicaten kunnen verdwijnen zonder dat de tabel wijzigt. Stel dat de systeemtijd 12 januari 1999 is. Rij 4 en 5 zijn op dat moment actuele duplicaten. Eén dag later, op 13 januari 1999, is de systeemtijd niet meer gemeenschappelijk in de geldigheidsintervallen en zijn deze twee rijen geen actuele duplicaten meer; het blijven echter intervalduplicaten.

| Aandeelnummer | Aandeelnaam | Start | Einde |
|---|---|---|---|
| … | … | … | … |
| 1001 | SCOB | 11/12/1995 | 03/04/1998 |
| 1001 | SCOB | 11/12/1995 | 03/04/1998 |
| 1001 | SCOB | 04/04/1999 | 31/12/9999 |
| 1002 | UIA-RUCA | 11/12/1995 | 12/01/1999 |
| 1002 | UIA-RUCA | 01/01/1999 | 31/12/1999 |
| … | … | … | … |

**Tabel 10:** temporele tabel *Aandeelnaam*.

Een mogelijke SQL-definitie voor Tabel 10 is terug te vinden in SQL-fragment 5.

```
CREATE TABLE AandeelNaam
( AandeelNummer INTEGER,
  AandeelNaam VARCHAR(80),
  Start   DATE,
  Einde   DATE)
```

**SQL-fragment 5:** definitie van de tabel *AandeelNaam*.

Waardeduplicaten zijn de zwakste vorm van temporele duplicaten vermits de geldigheidsduur buiten beschouwing wordt gelaten. In SQL kunnen waardeduplicaten vermeden worden door aan de tabeldefinitie een *UNIQUE*-clausule toe te voegen waarin alle niet-temporele attributen vermeld staan. De tabeldefinitie ziet er dan als volgt uit:

```
CREATE TABLE AandeelNaam
( AandeelNummer INTEGER,
  AandeelNaam VARCHAR(80),
  Start   DATE,
  Einde   DATE,
  UNIQUE (AandeelNummer, AandeelNaam))
```

**SQL-fragment 6:** vermijden van waardeduplicaten in de tabel *AandeelNaam*.

Voor de tabel *AandeelNaam* is het verbieden van waardeduplicaten niet echt realistisch vermits dit zou betekenen dat eens een aandeel van naam verandert het deze naam nooit meer mag aannemen terwijl het in realiteit meermaals voorkomt dat een aandeel een naam krijgt die het al eerder had.

Attribuutduplicaten kunnen worden vermeden door naast de niet-temporele attributen ook de attributen *Start* en *Einde* op te nemen in de *UNIQUE*-clausule, zie bv. SQL-fragment 7. Toch is dit niet echt zinvol want het verbieden van attribuutduplicaten kan eenvoudig omzeild worden door de waarde van het attribuut *Start* of *Einde* een fractie te wijzigen; hierdoor zijn niet meer alle attribuutwaarden identiek en is er bijgevolg geen sprake meer van attribuutduplicaten.

```
CREATE TABLE AandeelNaam
( AandeelNummer INTEGER,
  AandeelNaam VARCHAR(80),
  Start   DATE,
  Einde   DATE,
  UNIQUE (AandeelNummer, AandeelNaam, Start, Einde)
)
```

**SQL-fragment 7:** vermijden van attribuutduplicaten in de tabel *AandeelNaam*.

Het vermijden van intervalduplicaten gebeurt met *CHECK*-clausule in de tabeldefinitie; deze clausule controleert dan of er waardeduplicaten voorkomen die een overlappende geldigheidsduur hebben, als volgt:

```
CREATE TABLE AandeelNaam
( AandeelNummer INTEGER,
  AandeelNaam VARCHAR(80),
  Start   DATE,
  Einde    DATE,
  CHECK (NOT EXISTS(SELECT *
                    FROM AandeelNaam A
                    WHERE 2<=(SELECT COUNT(*)
                              FROM AandeelNaam B
                              WHERE A.AandeelSleutel=B.AandeelSleutel
                                AND A.AandeelNaam=B.AandeelNaam
                                AND A.Start<B.Einde
                                AND A.Einde>B.Start))
)
```

**SQL-fragment 8:** vermijden van intervalduplicaten in de tabel *AandeelNaam*.

Noot: De SQL-variant in Oracle8 ondersteunt de *CHECK*-clausule enkel wanneer er niet verwezen wordt naar tabellen; de code uit SQL-fragment 8 wordt dus niet ondersteund. Vermijden van intervalduplicaten in Oracle8 kan opgelost worden via SQL-*triggers*, de code hiervoor is terug te vinden in Appendix B.

Het vermijden van actuele duplicaten is een bizarre opdracht want zoals gezegd kunnen actuele duplicaten vanzelf verdwijnen; er blijven echter nog wel intervalduplicaten. In sommige gevallen is het nuttig om te eisen dat de temporele tabellen momentherleidbaar zijn op het tijdstip *heden*, hierdoor kan de databank op het tijdstip *heden* beschouwd worden als niet-temporeel.
Actuele duplicaten kunnen vermeden worden door in de *WHERE*-clausule van de binnenste *SELECT*-opdracht van SQL-fragment 8 deze lijnen toe te voegen:

```
AND A.Start<=SysDate AND SysDate<=A.Einde
AND B.Start<=SysDate AND SysDate<=B.Einde
```

## *4.5 Intervalmaximalisatie*

### 4.5.1 Definitie

Intervalmaximalisatie is een unaire operator (afgekort *IM*) die wordt toegepast op temporele tabellen waarbij intervalduplicaten worden *versmolten*[20] tot één tuppel met als geldigheidsduur de unie van de geldigheidsduur van de duplicaten. Dat de operator *IM* een absolute noodzaak is in temporele databanken, wordt aangetoond met volgend voorbeeld:

Onderstaande tabel stelt een temporele relatie voor die aanduidt wanneer een bepaald aandeel genoteerd staat op een beurs.

| Naam | Beurs | Levensduur |
|---|---|---|
| Kredietbank | Brussel | 01/01/1981 → 07/04/1989 |
| Cera | Brussel | 03/04/1985 → 03/09/1998 |
| Kredietbank | Brussel | 08/04/1989 → 31/12/1994 |

**Tabel 11:** niet-intervalgemaximaliseerde tabel.

Wanneer met een bevraging wordt bepaald welke aandelen ooit langer dan tien jaar op een beurs genoteerd stonden, dan zou de databank enkel *Cera* als resultaat geven, terwijl *Kredietbank* duidelijk ook langer dan 10 jaar genoteerd stond. Het probleem zit in het feit dat de databank niet automatisch in staat is om te ontdekken dat de twee periodes 1/1/1981 t.e.m. 7/4/1989 en 8/4/1989 t.e.m. 31/12/1994 elkaar direkt opvolgen en dus equivalent zijn met het interval 1/1/1981 t.e.m. 31/12/1994. Afzonderlijk zijn beide intervallen korter dan tien jaar maar tezamen bestrijken de intervallen een periode van dertien jaar.

[20] Vandaar dat intervalmaximalisatie in het Engels wordt uitgedrukt met *coalescing* wat samensmelting betekent.

Een tabel is niet-intervalgemaximaliseerd (afgekort: *niet-Im*) wanneer er intervalduplicaten in voorkomen. Het is belangrijk te weten dat bij intervalmaximalisatie geen informatie verloren gaat en geen nieuwe informatie wordt toegevoegd aan de tabel.
Meteen kan men zien dat de operator *IM* geen elementaire bewerking is; net als bij het verwijderen van duplicaten moet eerst gecontroleerd worden op de gelijkheid van alle attribuutwaarden, gevolgd door een overlappingstest van de intervallen. Tenslotte moeten alle overlappende intervallen samengesmolten worden en alle niet-maximale intervallen verwijderd.

Men zou kunnen opperen dat intervalmaximalisatie een overbodige operatie is vermits in paragraaf 4.4 een methode is besproken om intervalduplicaten a priori te vermijden in een tabel. Deze methode bespreekt echter alleen hoe *insert*- of *update*-opdrachten die mogelijk intervalduplicaten introduceren, onderschept kunnen worden. De onderschepte opdrachten worden gewoonweg niet uitgevoerd. Dit is niet altijd gewenst: soms is het belangrijker dat de gegevens wel gestockeerd worden ook al leidt dit tot intervalduplicaten. Intervalmaximalisatie elimineert deze duplicaten dan weer. Daarenboven zal worden aangetoond dat bepaalde tabeloperaties uit de temporele algebra de eigenschap *Im* niet bewaren. De operator *IM* moet dus opnieuw worden uitgevoerd op het resultaat.

### 4.5.2 Eigenschappen

Vermits de operator *IM* zeer kostelijk is, is het belangrijk om het aantal *IM*-operaties zo laag mogelijk te houden.
In [BOHL97] wordt eerst bekeken welke bewerkingen uit de temporele algebra de *Im*-eigenschap bewaren, daarna worden een paar nuttige eigenschappen van de operator *IM* bekeken.

**Stelling 1:** Temporele selectie bewaart de *Im*-eigenschap.

Bewijs:

Het resultaat van een temporele selectie is altijd een deelverzameling van de oorspronkelijke relatie en vermits de geldigheidsduur van de oorspronkelijke tuppels ongewijzigd blijft, blijft de *Im*-eigenschap gelden. □

**Stelling 2:** Temporele projectie bewaart de *Im*-eigenschap niet noodzakelijk.

Bewijs met tegenvoorbeeld:

Neem het relatieschema $R_1=(A,B \,||\, [S \rightarrow E])$ met volgende intervalgemaximaliseerde instantie:
$r=\{<a,b||[2 \rightarrow 5]>,<a,c \,||\, [5 \rightarrow 8]>\}$
$\pi_{t,A}\, r=\{<a \,||\, [2 \rightarrow 5]>,<a \,||\, [5 \rightarrow 8]>\}$
De resultaatverzameling is duidelijk niet *Im*. □

**Stelling 3:** Temporele unie bewaart de *Im*-eigenschap niet noodzakelijk.

Bewijs met tegenvoorbeeld:

Neem de relatieschema's $R_1=(A,B \,||\, [S \rightarrow E])$ en $R_2=(A,B \,||\, [S \rightarrow E])$ met volgende interval-gemaximaliseerde instanties: $r_1=\{<a,b||[2 \rightarrow 5]>,<a,c||[5 \rightarrow 8]>\}$ en $r_2=\{<a,b||[5 \rightarrow 9]>\}$
$r_1 \cup_t r_2 = \{<a,b||[2 \rightarrow 5]>,<a,c||[5 \rightarrow 8]>,<a,b||[5 \rightarrow 9]>\}$
De resultaatverzameling is duidelijk niet *Im*. □

**Stelling 4:** Temporeel carthesisch product behoudt *Im*.

Bewijs:

Het temporeel carthesisch product kan beschouwd worden als een uitbreiding van de attributen van tuppels in $r_1$, gevolgd door een selectie in $r_1$ en parallel daarmee eventueel een inkorting van de geldigheidsduur van de tuppels. Geen van deze operaties kan leiden tot het verlies van *Im*. □

**Stelling 5:** Het temporeel verschil behoudt *Im*.

Bewijs:

De geldigheidsduur van tuppels uit $r_1$ kan enkel ingekort worden dus blijft de *Im*-eigenschap bewaard. □

Een zeer eenvoudige regel vermijdt de opeenvolging van *IM*-operaties:

- *IM1: IM(IM(r)) = IM(r)*

Vermits temporele selectie, temporeel carthesisch product en temporeel verschil de *Im*-eigenschap bewaren, gelden volgende regels:

- *IM2:* $IM(\sigma_{v,c}(IM(r)) = \sigma_{v,c}\ IM(r)$
- *IM3:* $IM(r_1\ x_t\ r_2) = IM(r_1)\ x_t\ IM(r_2)$
- *IM4:* $IM(r_1 \setminus_t r_2) = IM(r_1) \setminus_t IM(r_2)$

De regels *IM3* en *IM4* bepalen echter niet in welk lid van de gelijkheid de operator *IM* het snelst wordt uitgevoerd. Dit moet op *run-time* bepaald worden; intuïtief is het efficiënter om de operator *IM* slechts één keer uit te voeren, dus om de linkerleden van gelijkheden te gebruiken maar dit geen wet.

Voor het temporeel verschil geldt daarenboven nog:

- *IM5:* $r_1 \setminus_t IM(r_2) = r_1 \setminus_t r_2$

Hoewel de temporele unie de *Im*-eigenschap niet bewaart, kan ook hier het aantal IM-operaties beperkt worden:

- *IM6:* $IM(IM(r_1) \cup_t IM(r_2)) = IM(r_1 \cup_t r_2)$

De regels *IM1* t.e.m. *IM6* gelden onder alle omstandigheden. Indien te achterhalen is of een temporele operatie aan bepaalde voorwaarden voldoet dan zijn er nog een paar meer specifieke regels afleidbaar; deze gelden uiteraard niet meer onder alle omstandigheden.

Indien een DBMS beschikt over een predikaat *is_IM(t)* dat achterhaalt of een bepaalde tabel *t Im* is dan wordt volgende regel mogelijk:

- *IM7: IM(r)=r* $\Leftrightarrow$ *is_IM(r) of wanneer* r *a priori* Im *is*

*IM7* is bijzonder nuttig in databanken die een *im*-vlag bijhouden voor elke tabel. Wanneer bij controle van deze vlag blijkt dat een tabel *t Im* is, dan kan de opdracht *IM(t)* volledig genegeerd worden.

Wanneer bij de selectieconditie geen rekening wordt gehouden met de geldigheidsduur van de oorspronkelijke tuppels, dan kunnen de operatoren $\sigma$ en *IM* omgewisseld worden.

- *IM8:* $\sigma_{v,cond}(IM(r)) = IM(\sigma_{v,cond}(r))$ $\Leftrightarrow$ *cond is onafhankelijk van geldigheidsduur*

Het nut van *IM8* wordt duidelijk wanneer de $\sigma$-operator het merendeel van de tuppels elimineert.

Een analoge regel geldt voor de temporele projectie:

- *IM9:* $IM(\pi_{v,f}(IM(r))) = IM(\pi_{v,f}(r))$ $\Leftrightarrow$ f *is onafhankelijk van geldigheidsduur*

### 4.5.3 Intervalmaximalisatie in SQL

Een rechttoe-rechtaan algoritme overloopt iteratief de relatie en voegt waar mogelijk telkens tuppels samen. Dit wordt in twee SQL-stappen geïmplementeerd als volgt:

1. Zoek twee aan twee alle intervalduplicaten *a* en *b*. Creëer een derde intervalduplicaat *c* met $[S_c \rightarrow E_c]=[S_a \rightarrow E_a]\ U\ [S_b \rightarrow E_b]$ en voeg dit toe aan de relatie indien het nog niet voorkomt.

```
INSERT INTO r
SELECT a.A, a.S, b.E
FROM r a, r b
WHERE a.A=b.A
  AND a.S<b.S
  AND b.S<=a.E
  AND a.E<b.E
MINUS SELECT * FROM r
```

**SQL-fragment 9:** eerste stap in de iteratieve *IM*-implementatie.

De eerste regel van de *WHERE*-clausule eist dat de tuppels waardeduplicaten zijn, de drie resterende regels garanderen dat de geldigheidsduur van *b* later in de tijd overlapt met de geldigheidsduur van *a*. De *MINUS*-clausule vermijdt attribuutduplicaten in *r*.
Deze stap wordt geïtereerd zolang de tabel verandert, dus zolang er tuppels worden toegevoegd.

2. Verwijder alle intervalduplicaten waarvoor de geldigheidsduur niet maximaal is.

```
DELETE FROM r a
WHERE EXISTS (SELECT *
              FROM r b
              WHERE a.A=b.A
                AND a.S>b.S  AND a.E<=b.E
                 OR a.S>=b.S AND a.E<b.E)
```

**SQL-fragment 10:** tweede stap in de iteratieve *IM*-implementatie.

De operator *IM* kan ook worden uitgedrukt in één SQL-statement dat gebaseerd is op volgende formule uit eerste-orde-logica die de *Im*-eigenschap uitdrukt:

IM: $r(<X \;||\; [S\rightarrow\_]>) \wedge r(<X \;||\; [\_\rightarrow E]>) \wedge S<E \wedge$
$\forall A \bullet (r(<X \;||\; [A\rightarrow\_]>) \wedge S<A<E \Rightarrow \exists U,V\bullet(r(<X \;||\; [U\rightarrow V]>) \wedge U<A\leq V)) \wedge$
$\neg\exists A,B \bullet(r(<X \;||\; [A\rightarrow B]>)\wedge(A<S\leq B \vee A\leq E<B))$

Lijn 1 selecteert twee (mogelijk identieke) waardeduplicaten die de geldigheidsduur bepalen van een interval-gemaximaliseerd tuppel.
Lijn 2 verzekert dat alle waardeduplicaten waarvan de geldigheidsduur begint na *S* en voor *E*, worden uitgebreid door een ander waardeduplicaat. Er kunnen aldus geen gaten zitten in het interval *[S→E]*.
Lijn 3 garandeert dat het resultaat de grootst mogelijke geldigheidsduur heeft; *[S→E]* mag dus geen deel uitmaken van een grotere geldigheidsduur van een waardeduplicaat.

In lijn 2 kunnen de universele kwantor en de implicatie worden weggewerkt door volgende equivalenties:
$\forall A\bullet(p(A)\Rightarrow q(A)) \;\Leftrightarrow\; \neg\exists A\bullet\neg(p(A)\Rightarrow q(A)) \;\Leftrightarrow\; \neg\exists A\bullet p(A)\wedge\neg q(a)$

Lijn 2 wordt dan: $\neg\exists A\bullet(r(<X \;||\; [A\rightarrow\_]>)\wedge S<A<E \wedge \neg\exists U,V\bullet(r(<X \;||\; [U\rightarrow V]>) \wedge U<A\leq V))$
De hele formule kan nu omgezet worden naar een SQL-bevraging, nl:

```
SELECT DISTINCT a.A, a.S, b.E
FROM r a, r b
WHERE a.S<=b.E
  AND a.A=b.A
  AND NOT EXISTS (SELECT *
                  FROM r c
                  WHERE a.A=c.A
                    AND a.S<c.S AND c.S<b.E
                    AND NOT EXISTS (SELECT *
                                    FROM r d
                                    WHERE a.A=d.A
                                      AND d.S<c.S AND c.S<=d.E
                                   )
                 )
  AND NOT EXISTS (SELECT *
                  FROM r e
                  WHERE e.A=a.A
                    AND (    e.S<a.S AND a.S<=e.E
                          OR e.S<=b.E AND b.E<e.E
                        )
                  )
```

**SQL-fragment 11:** *IM*-implementatie in één SQL-opdracht.

Het resultaat van deze bevraging is de *Im*-tabel. De eigenlijke tabel blijft ongewijzigd, dit in tegenstelling met de iteratieve methode. SQL-fragment 11 moet slechts eenmaal worden uitgevoerd om tabel *t Im* te maken. Het valt meteen op dat deze SQL-opdracht een stuk complexer is dan de twee opdrachten uit de iteratieve methode. In [BOHL97] wordt empirisch aangetoond dat de niet-iteratieve methode nooit efficiënter presteert dan de iteratieve; de oorzaak hiervan is dat de winst die wordt behaald doordat de SQL-opdracht slechts eenmaal moet worden uitgevoerd, volledig wordt tenietgedaan door de extreem hoge complexiteit van SQL-fragment 11.

### 4.5.4 Intervalmaximalisatie in de SCOB-databank

De noodzaak van intervalmaximalisatie in de SCOB-databank kan worden aangetoond aan de hand van de invoerstrategie. De hoeveelheid gegevens is zo gigantisch dat meerdere personen tegelijk gegevens zullen invoeren. Voor koersgegevens is het onmogelijk dat deze personen data invoeren uit dezelfde periode. Hoofdoorzaak hiervoor is dat de gegevens zijn opgenomen in zware onhandelbare koersboeken die amper door één persoon gehanteerd kunnen worden, laat staan door meerdere. Verschillende invoerders zullen dus gegevens ingeven uit verschillende periodes.

Stel dat invoerder *A* koersen ingeeft vanaf 1/1/1860 t.e.m. 31/12/1869. Soms wordt er een nieuw aandeel geïntroduceerd in de koerslijsten. In een complexere versie van Tabel 11 (p. 39), wordt dan een tuppel toegevoegd dat aangeeft dat het nieuwe aandeel vanaf dag *d* genoteerd staat op de beurs. De geldigheidsduur van de notering is nu gelijk aan *[d→d]*. Telkens wanneer gebruiker *A* overgaat naar een volgende koersdag wordt de levensduur van de notering automatisch met één dag verlengd, tenzij de gebruiker expliciet aangeeft dat deze verlenging niet meer mag gebeuren omdat het aandeel van de beurs geschrapt is.
Wanneer *A* klaar is met de periode *[1/1/1860→31/12/1869]* dan hebben alle door *A* ingegeven noteringen (behalve geschrapte aandelen) een levensduur *[S→31/12/1869]* met $S \in [1/1/1860 \rightarrow 31/12/1869]$. Ondertussen is invoerder *B* bezig met de periode *[1/1/1870→31/12/1879].* Alle door *B* ingegeven noteringen hebben dus een levensduur *[1/1/1870→E]* met $E \in [1/1/1870 \rightarrow 31/12/1879]$.
Een aandeel *X* dat zowel genoteerd staat in 1869 als in 1870 komt dus twee keer voor in de tabel *Notering* namelijk:

`Notering(X,[S→31/12/1869])` en `Notering(X,[01/01/1870→E])`

Wanneer *A* en *B* gereed zijn met hun invoerwerk dan moet de operator *IM* worden uitgevoerd op de tabel *Notering*.

Men zou kunnen opperen dat invoerder *A* en *B* twee periodes moeten ingeven die zo ver uit elkaar liggen dat de kans op overlappende noteringen bijzonder klein wordt; hierdoor zou de intervalmaximalisatie manueel uitgevoerd kunnen worden. Maar dit is praktisch onhaalbaar: niet alleen de koersinformatie moet worden ingegeven, ook de structuur van de beurs, informatie over de bedrijven die het aandeel uitgaven, informatie over leidinggevende personen van die bedrijven en nog heel wat bijkomende informatie. Wanneer de invoerperiodes voor *A* en *B* vrij dicht bij elkaar liggen dan kan de hoeveelheid bijkomende in te voeren informatie beperkt worden.

### 4.5.5 Variatie op intervalmaximalisatie in de SCOB-databank

De operator *IM'* vervangt *n* attribuutduplicaten tuppels $<X \,||\, [S_i \rightarrow E_i]>$ met $1 \leq i \leq n$ vervangen door het tuppel $<X \,||\, min(S_1, \ldots S_n) \rightarrow max(E_1, \ldots E_n)]>$.
De operator *IM'* is niet geschikt voor de geldigheidsduur van attributen omdat dan nieuwe eventueel foutieve informatie wordt toegevoegd maar IM' is wel geschikt voor de levensduur van entiteiten waarvan die betreffende levensduur een ononderbroken tijdsinterval moet zijn.

Ter illustratie de personenmodule uit de SCOB-databank: sinds 1830 hebben heel wat personen een belangrijke rol gespeeld in de evolutie van het beurs- en bedrijfswezen. Van heel veel van die personen zijn de geboorte- en overlijdensdatum niet juist gekend. Toch zijn juist deze twee datums nodig om de levensduur van een *PERSOON*-entiteit te bepalen. Wanneer de twee datums wel bekend zijn dan is er geen probleem. Als ze niet gekend zijn dan kan men volgende methode gebruiken. Stel dat van *Jan Jansen* enkel het volgende bekend is:

- van 3/4/1875 t.e.m. 21/9/1889 was hij eigenaar van de firma Jansen
- van 1/1/1893 t.e.m. 14/2/1895 was hij gehuwd met Els Elzen

Wanneer de invoerder *Jan Jansen* toevoegt aan de databank dan kan hij de levensduur met zekerheid al vastleggen op [3/4/1875→14/2/1895], d.i. het kleinste interval dat zowel [3/4/1875→21/9/1889] als [1/1/1893→14/2/1895] omvat. Ondertussen geeft de invoerder nog honderden andere personen in, zonder telkens te controleren of de ingegeven personen al voorkomen in de databank. Op een bepaald moment moet hij ingeven dat ene *Jan Jansen* van [6/7/1904→31/12/1913] voorzitter was van een Raad van Beheer.
De levensduur van entiteit *Jan Jansen* kan worden verlengd tot en met 31/12/1913, hoewel hij nog geen informatie heeft die aantoont dan *Jan Jansen* leefde tussen 14/2/1895 en 6/7/1904. Vermits een persoon niet tijdelijk kan sterven, hoeft de gebruiker niet over deze informatie te beschikken.

Zowel de IM- als de IM'-operator zullen belangrijk zijn voor het onderhoud van de SCOB-databank want de invoerder kan onmogelijk bij elke *insert*- of *update*-opdracht op een temporeel attribuut controleren of de geldigheidsduur van het attribuut effectief een deelinterval is van de levensduur van de betrokken entiteit.

## *4.6 Temporele kardinaliteiten in SQL*

Het TER-model introduceerde ontdubbelde temporele kardinaliteiten (paragraaf 2.2.5); deze zijn overgenomen in het LACER-model (paragraaf 3.6) en komen veelvuldig voor in het LACER-diagramma van de SCOB-databank (Appendix A). Bij de omzetting van een LACER-diagramma naar een relationeel schema moeten deze temporele kardinaliteiten gegarandeerd blijven. Deze omzetting wordt naar mijn mening serieus onderschat want ik heb er is zo goed als geen valabele literatuur over gevonden. Er wordt veelal voor gekozen om de kardinaliteiten niet te garanderen maar slechts op geregelde tijdstippen een bevraging uit te voeren die alle aanwezige relatie-instanties opsomt die niet voldoen aan de geëiste kardinaliteiten.

Toch wil ik een overzicht geven van hoe temporele kardinaliteiten wel gegarandeerd kunnen worden. Dit kan ondermeer gebeuren via *CHECK*-clausules in de tabeldefinities. Het is echter ondoenbaar om elke mogelijke temporele kardinaliteit te bespreken, daarom wordt deze bespreking beperkt tot temporele attribuut-kardinaliteiten. De temporele kardinaliteiten van een binaire relatie kunnen beschouwd worden als twee attribuutkardinaliteiten van een relatie-entiteit.
In deze paragraaf wordt gewerkt met twee tabellen *A* en *B*, waarbij *A* een temporele entiteit voorstelt met een temporeel attribuut *B*. De implementatie van *A* en *B* in SQL ziet er dan als volgt uit:

```
CREATE TABLE A
( A_Sleutel INTEGER PRIMARY KEY,
  Start DATE,
  Einde DATE)

CREATE TABLE B
( B_Sleutel INTEGER PRIMARY KEY ,
  A_Sleutel INTEGER REFERENCES A(A_Sleutel),
  B_Waarde  gelijk_welk_type,
  Start DATE,
  Einde DATE)
```

**SQL-fragment 12:** creatie van de voorbeeldtabellen *A* en *B*.

Er is gekozen om voor beide tabellen een loze sleutel in te voeren, dit maakt het opstellen van de beperkingen op de tabellen eenvoudiger.

Een algemene temporele beperking is dat de geldigheidsduur van elke *B*-waarde een deelinterval moet zijn van de levensduur van de bijbehorende *A*-entiteit. Deze beperking kan op twee manieren geschonden worden:

- een *update*-opdracht op *A* waarbij de levensduur wordt ingekort. Deze schending kan worden opgevangen door de opdracht te weigeren ofwel door de geldigheidsduur van alle compromitterende *B*-waarden in te korten en eventueel *B*-waarden te verwijderen uit de tabel.
- een *insert*- of *update*-opdracht op *B* waarbij een tuppel ontstaat waarvan de geldigheidsduur buiten de levensduur van *A* valt. In dit geval kan de modificatie ook geweigerd worden ofwel kan de levensduur van de bijbehorende *A*-entiteit verlengd worden of de geldigheidsduur van *B* ingekort.

Het weigeren van een opdracht kan gebeuren via eenvoudige *triggers* of *CHECK*-clausules; het automatisch aanpassen van de geldigheidsduur of levensduur in de andere tabel is geen eenvoudige opdracht vermits niet elke compromitterende opdracht kan omgezet worden. Daarenboven is het meestal niet gewenst dat dergelijke opdrachten daadwerkelijk worden uitgevoerd vermits ze meestal inconsistent zijn; door de oorspronkelijke tabellen aan te passen wordt deze inconsistentie verspreid doorheen de tabellen. In deze paragraaf wordt dan ook gekozen om compromitterende opdrachten te weigeren.

De algemene temporele beperking wordt bewerkstelligd door een *CHECK*-clausule toe te voegen aan beide tabeldefinities. Deze clausule controleert of de levensduur van een entiteit *a* altijd de geldigheidsduur van alle bijbehorende attributen *b* omvat. De clausule is voor beide tabellen identiek:

```
CHECK (NOT EXISTS(SELECT *
                  FROM A
                  WHERE EXISTS(SELECT *
                               FROM B
                               WHERE B.A_Sleutel=A.Sleutel
                                 AND (B.Begin<A.Begin OR B.Einde>A.Einde))))
```

**SQL-fragment 13:** garantie van de algemene temporele beperking.

Wat volgt is een overzicht van attribuutkardinaliteiten en hoe ze bewerkstelligd kunnen worden. Meestal geldt de SQL-implementatie voor meerdere kardinaliteiten, vandaar dat enkel de relevante elementen van de kardinaliteit zijn ingevuld, de irrelevante worden voorgesteld door een heksymbool: #.

Met de implementatie van de algemene temporele beperking is meteen ook de attribuutkardinaliteit *[0,0,M,N]* bewerkstelligd, m.a.w. deze kardinaliteit behoeft geen verdere beperking op de tabellen *A* en *B*.

De attribuutkardinaliteit *[#,#,1,#]* kan verwezenlijkt worden door een controle die lijkt op het vermijden van intervalduplicaten, het enige verschil is dat nu niet meer gecheckt wordt op alle attributen van *B*. Geen enkel van de tuppels met eenzelfde attribuut *A_Sleutel* mag een overlappende geldigheidsduur hebben. Dit wordt met volgende *CHECK*-clausule in tabel *B* gegarandeerd:

```
CHECK(NOT EXISTS(SELECT *
                 FROM B B1
                 WHERE 2<=(SELECT COUNT(*)
                           FROM B B2
                           WHERE B1.A_Sleutel=B2.A_Sleutel
                             AND B1.Begin<B2.Einde
                             AND B2.Begin<=B1.Einde)))
```

**SQL-fragment 14:** garantie van de temporele kardinaliteit *[#,#,1,#]*.

De attribuutkardinaliteit *[#,1,#,#]* is te bewerkstelligen met volgende *CHECK*-clausule in beide tabellen:

```
CHECK (NOT EXISTS (SELECT *
                   FROM A
                   WHERE NOT EXISTS (SELECT *
                                     FROM B
                                     WHERE A_Sleutel=A.Sleutel)))
```

**SQL-fragment 15:** garantie van de temporele kardinaliteit *[#,1,#,#]*.

Wanneer de kardinaliteit niet *[#,1,#,#]* is, maar *[#,x,#,#]* met *x>1* dan wordt volgende *CHECK*-clausule gebruikt:

```
CHECK (NOT EXISTS (SELECT *
                   FROM A
                   WHERE x>(SELECT COUNT(*)
                            FROM B
                            WHERE A_Sleutel=A.Sleutel)))
```

**SQL-fragment 16:** garantie van de temporele kardinaliteit *[#,x,#,#]*.

De eis tot momentverplichtheid uit zich in de attribuutkardinaliteit *[1,#,#,#]*. In temporele algebra moet volgende formule voldaan zijn: $\pi_{t,Sleutel}\ A \setminus_t \pi_{t,A\text{-}Sleutel}\ B = \varnothing$
Dit kan niet zomaar omgezet worden in een *CHECK*-clausule.
Een mogelijke oplossing is:

- Voer de bevraging `SELECT A_Sleutel, Begin, Einde FROM B` uit.
  Dit komt overeen met $\pi_{t,A\text{-}Sleutel}\ B$
- Pas intervalmaximalisatie toe op de resultaattabel van voorgaande bevraging.
- Controleer voor elke *Sleutel*-waarde uit *A* of diens levensduur exact samenvalt met het bijbehorende gemaximaliseerde interval. Indien dit niet het geval is, dan is aan de moment-verplichtheid niet voldaan.

Wanneer de kardinaliteit niet *[1,#,#,#]* is maar *[x,#,#,#]* met *x>1* wordt het probleem nog complexer omdat intervalmaximalisatie hier geen uitkomst meer biedt. Men kan een temporeel attribuut $B_1$ toevoegen aan *A* met kardinaliteit *[1,#,#,#]* en tegelijk de kardinaliteit van *B* vervangen door *[x-1,#,#,#];* dit wordt dan x-1 maal herhaald zodat er enkel nog kardinaliteiten van de vorm *[1,#,#,#]* voorkomen. Dit is geen elegante oplossing want als beide kardinaliteiten voldaan zijn, waar moet dan een nieuwe attribuutwaarde *B* worden toegevoegd? Intuïtief is duidelijk dat dit best kan gebeuren in de tabel *B* maar dit is geen verplichting, de gebruiker kan en mag de nieuwe waarde ook toevoegen aan één van de nieuwe tabellen $B_i$. Bij een hoge waarde voor *x* zouden te veel extra tabellen ontstaan, hetgeen bevragingen over de oorspronkelijke tabel *B* bijzonder ingewikkeld maakt.

## *4.7 Temporele bevragingen in SQL*

In deze paragraaf worden bewust geen tabellen uit de SCOB-databank gebruikt en wel om twee redenen: ten eerste zou de structuur van de tabellen te veel inzicht vereisen in het beursgebeuren en ten tweede zijn die tabellen reeds aangepast voor efficiënte opslag waarbij de link met de SCOB-miniwereld niet altijd even duidelijk is. Daarom neem ik de voorbeeldtabel uit [SNOD98]: er worden gegevens bijgehouden over koeienloten in stallen.

| Lot | Stal | Aantal | Start | Einde |
|---|---|---|---|---|
| 137 | 1 | 17 | 07/02/1998 | 18/02/1998 |
| 219 | 1 | 43 | 25/02/1998 | 28/02/1998 |
| 219 | 1 | 20 | 01/03/1998 | 13/03/1998 |
| 219 | 2 | 23 | 01/03/1998 | 13/03/1998 |
| 219 | 2 | 43 | 14/03/1998 | 31/12/9999 |
| 374 | 1 | 14 | 20/02/1998 | 31/12/9999 |
| … | … | … | … | … |

**Tabel 12:** de tabel *LotPerStal*.

Er zijn drie categorieën bevragingen mogelijk op deze tabel:

- *actuele bevraging*: deze geeft hetzelfde resultaat als zou het hier gaan over een actuele tabel.
  *Hoeveel koeien uit lot 219 bevinden zich momenteel in elke stal?*

```
SELECT Stal, Aantal
FROM LotPerStal
WHERE Lot=219
  AND Einde=DATE '31/12/9999'
```

| Stal | Aantal |
|---|---|
| 2 | 43 |

- *waardebevraging*: deze doorloopt alle rijen zonder rekening te houden met de geldigheidsduur. *Hoeveel koeien uit lot 219 bevonden zich ooit in elke stal?*

```
SELECT Stal, Aantal
FROM LotPerStal
WHERE Lot=219
```

| Stal | Aantal |
|---|---|
| 1 | 43 |
| 1 | 20 |
| 2 | 23 |
| 2 | 43 |

  Het resultaat van waardebevraging is niet altijd even duidelijk maar uit deze tabel valt bijvoorbeeld af te leiden dat er 4 verplaatsingen zijn gebeurd met koeien uit lot 219.

- *intervalbevraging*: dit is waardebevraging waarbij de geldigheidsduur wel in rekening wordt gebracht.
  *Hoeveel koeien uit lot 219 bevonden zich ooit in elke stal?*

```
SELECT Stal, Aantal, Start, Einde
FROM LotPerStal
WHERE Lot=219
```

| Stal | Aantal | Start | Einde |
|---|---|---|---|
| 1 | 43 | 25/02/1998 | 01/03/1998 |
| 1 | 20 | 01/03/1998 | 14/03/1998 |
| 2 | 23 | 01/03/1998 | 14/03/1998 |
| 2 | 43 | 13/03/1998 | 31/12/9999 |

De Engelstalige benaming voor actuele, waarde- en intervalbevraging zijn resp. *current*, *nonsequenced* en *sequenced querying*.
Het verschil tussen waarde- en intervalbevraging is misschien nog onduidelijk maar bij bevragingen met *joins* is het verschil veel groter.

Tot dit punt zijn de temporele bevragingen nog rechttoe-rechtaan maar wanneer we volgende vraag willen stellen worden de zaken ingewikkelder: «*Wanneer bevonden zich verschillende loten in dezelfde stal?*»

In een actuele tabel is deze bevraging nog eenvoudig:

```
SELECT A.Lot, B.Lot, A.Stal
FROM LotPerStal A, LotPerStal B
WHERE A.Lot<B.Lot
  AND A.Stal=B.Stal
```

In de temporele tabel moet gezocht worden naar alle loten *A* en *B* die zich in dezelfde stal bevinden tijdens een gemeenschappelijk tijdsinterval dus naar loten in eenzelfde stal waarvan de intersectie van de geldigheidsduur niet leeg is. Hiervoor zijn vier mogelijkheden:

- De geldigheidsduur van *A* is een deelinterval van de geldigheidsduur van *B*.
  $S_A \geq S_B \wedge E_A \leq E_B \Rightarrow [S_A \rightarrow E_A] \cap [S_B \rightarrow E_B] = [S_A \rightarrow E_A]$
- De geldigheidsduur van *B* is een deelinterval van de geldigheidsduur van *A*.
  $S_A \leq S_B \wedge E_A \geq E_B \Rightarrow [S_A \rightarrow E_A] \cap [S_B \rightarrow E_B] = [S_B \rightarrow E_B]$
- De geldigheidsduur van *A* is nog bezig als die van *B* begint.
  $S_A < S_B \wedge E_A < E_B \wedge E_A > S_B \Rightarrow [S_A \rightarrow E_A] \cap [S_B \rightarrow E_B] = [S_B \rightarrow E_A]$
- De geldigheidsduur van *B* is nog bezig als die van *A* begint.
  $S_A > S_B \wedge E_A > E_B \wedge S_A < E_B \Rightarrow [S_A \rightarrow E_A] \cap [S_B \rightarrow E_B] = [S_A \rightarrow E_B]$

In elk van deze gevallen is telkens een andere conditie voldaan zijn en moet een ander interval geselecteerd worden. In standaard SQL[21] is dit niet mogelijk in één opdracht dus moeten de vier gevallen afzonderlijk opgevraagd en samengevoegd worden tot één resultaat, als volgt:

```
SELECT A.Lot, B.Lot, A.Stal, A.Begin, A.Einde
FROM LotPerStal A, LotPerStal B
WHERE A.Lot<B.Lot
  AND A.Stal=B.Stal
  AND A.Begin>=B.Begin
  AND A.Einde<=B.Einde
UNION
SELECT A.Lot, B.Lot, A.Stal, B.Begin, B.Einde
FROM LotPerStal A, LotPerStal B
WHERE A.Lot<B.Lot
  AND A.Stal=B.Stal
  AND A.Begin<=B.Begin
  AND A.Einde>=B.Einde
UNION
SELECT A.Lot, B.Lot, A.Stal, B.Begin, A.Einde
FROM LotPerStal A, LotPerStal B
WHERE A.Lot<B.Lot
  AND A.Stal=B.Stal
  AND A.Begin<B.Begin
  AND A.Einde<B.Einde
  AND A.Einde>B.Begin
UNION
SELECT A.Lot, B.Lot, A.Stal, A.Begin, B.Einde
FROM LotPerStal A, LotPerStal B
WHERE A.Lot<B.Lot
  AND A.Stal=B.Stal
  AND A.Begin>B.Begin
  AND A.Einde>B.Einde
  AND A.Begin<B.Einde
```

**SQL-fragment 17:** intervalbevraging met *joins*.

Via de *UNION*-opdracht worden de vier resultaattabellen samengevoegd tot één resultaat zonder duplicaten, vermits *UNION* automatisch alle duplicaten verwijdert. Nadeel hierbij is dat bij elke *UNION*-opdracht duplicaten verwijderd worden, wat een overbodige taak is want we kunnen exact achterhalen wanneer er duplicaten kunnen optreden.

[21] Appendix B toont hoe deze bevraging efficiënter in Oracle8 PL/SQL kan opgesteld worden.

Dit kan namelijk enkel wanneer de geldigheidsduur van *A* en *B* exact samenvallen: zowel de eerste als de tweede *SELECT*-opdracht selecteren dan *A* en *B* met eenzelfde interval. Dit kan vermeden worden door bijvoorbeeld aan de *WHERE*-clausule van de tweede *SELECT*-opdracht het volgende toe te voegen:

```
AND NOT (A.Begin=B.Begin AND A.Einde=B.Einde)
```

Enkel de eerste *SELECT*-opdracht selecteert nog loten met gelijke geldigheidsduur. Omdat duplicaten nu a priori zijn uitgesloten, kunnen de drie *UNION*-opdrachten vervangen worden door *UNIONALL*-opdrachten die geen duplicaten elimineren en daardoor efficiënter zijn.

Het resultaat van de bevraging (eventueel met de vermelde aanpassingen) is:

| Lot | Lot | Stal | Start | Einde |
|---|---|---|---|---|
| 219 | 374 | 1 | 25/02/1998 | 28/02/1998 |
| 219 | 374 | 1 | 01/03/1998 | 14/03/1998 |

Dit resultaat is correct maar niet intervalgemaximaliseerd. Wanneer we intervalmaximalisatie toepassen op de tabel, dan is het uiteindelijke resultaat:

| Lot | Lot | Stal | Start | Einde |
|---|---|---|---|---|
| 219 | 374 | 1 | 25/02/1998 | 14/03/1998 |

Om dit resultaat te bekomen moesten vier *SELECT*-opdrachten uitgevoerd en samengevoegd worden, gevolgd door intervalmaximalisatie. Hiermee is duidelijk aangetoond dat bevraging van temporele tabellen geen vanzelfsprekende onderneming is.

De vorige bevraging was een intervalbevraging, de analoge waardebevraging klinkt als: *Welke loten hebben ooit in dezelfde stal gestaan, desnoods op verschillende tijdstippen?*
Deze bevraging kan uitgevoerd worden met volgende *SELECT*-opdracht:

```
SELECT UNIQUE A.Lot, B.Lot, A.Stal
FROM LotPerStal A, LotPerStal B
WHERE A.Lot<B.Lot
  AND A.Stal=B.Stal
```

Het resultaat van deze bevraging met de gegevens uit Tabel 12 is:

| Lot | Lot | Stal |
|---|---|---|
| 137 | 219 | 1 |
| 137 | 374 | 1 |
| 219 | 374 | 1 |

Loten 137 en 219 hebben nooit tegelijk in dezelfde stal gestaan maar ze hebben wel beide ooit in stal 1 gestaan.

Uiteraard zijn koeien geen aandelen en stallen geen beurzen maar in de SCOB-databank is een analoge bevraging niet ver te zoeken: «*Geef alle staalbedrijven die op een gemeenschappelijke beurscategorie aandelen hebben uitgegeven»*. Deze bevraging zal in een later onderzoeksstadium zeker uitgevoerd worden maar nogmaals, de modellering van aandeelnoteringen op een beurs is te ingewikkeld om als voorbeeld te dienen.

## *4.8 Temporele SQL*

In de voorgaande paragrafen is gebleken dat SQL niet echt geschikt is voor de behandeling van temporele tabellen:

- Duplicaateliminatie moet gebeuren via *CHECK*-clausules of *SQL*-triggers.
- Een *join* tussen temporele tabellen gebeurt aan de hand van de unie van vier bevragingen.
- Het garanderen van temporele kardinaliteiten gebeurt via meerdere *CHECK*-clausules per tabel.
- Intervalmaximalisatie gebeurt niet automatisch en de SQL-implementatie ervan heeft een hoge complexiteit.
- Automatische transactielogging vereist drie verschillende SQL-*triggers* per tabel.

Het probleem is dat SQL geen notie van tijd heeft en dus ook geen ingebouwde syntax of semantiek omvat voor temporele aspecten. In [SNOD97] en [SNOD98] wordt een temporele SQL (afgekort TSQL) voorgesteld waarbij het onderhoud van temporele tabellen wordt overgebracht van gebruiker naar DBMS. Kort gezegd komt het erop neer dat elke SQL-constructie wordt uitgebreid met een ondersteuning voor zowel geldigheidsduur als transactietijd en dit zonder dat de gebruiker meer kennis moet hebben over de interne structuur van deze uitbreidingen. TSQL-bevragingen worden hierdoor niet alleen korter maar ook transparanter en minder vatbaar voor gebruikersfouten.
In deze paragraaf wordt TSQL enkel illustratief uitgewerkt met de klemtoon op geldigheidsduur-tabellen; er wordt dus niet gestreefd naar volledigheid.

Een uitbreiding van SQL naar TSQL moet zo gebeuren dat de syntax noch de semantiek van SQL veranderen. Dit is een strenge eis die zich uit in volgende eigenschappen:

- *opwaartse compatibiliteit*: de huidige SQL-standaard moet integraal ondersteund worden door TSQL. Dus een SQL-databank (inclusief *triggers*, *views* en bevragingen) moet onveranderd blijven gelden in TSQL.
- *temporele opwaartse compatibiliteit*: wanneer een tabel temporeel wordt uitgebreid, moet een bestaande SQL-bevraging blijven gelden. Een SQL-bevraging van een temporeel uitgebreide tabel zal geïnterpreteerd worden als een actuele bevraging.
- *dubbele tijdsnotie*: TSQL moet zowel transactietijd als geldigheidsduur kunnen ondersteunen, desnoods tegelijkertijd.
- *algemene intervalondersteuning*: de temporele uitbreidingen moeten ondersteund worden door alle TSQL-constructies: bevragingen, modificaties, *views*, *triggers*, *constraints*…

Een elementaire uitbreiding in TSQL is het datatype *PERIOD(…)* waarbij tussen de haakjes vermeld wordt met welke tijdseenheid de grenzen van het tijdsinterval worden uitgedrukt.

- *PERIOD(DATE)*: dagniveau
- *PERIOD(TIME)*: secondeniveau
- *PERIOD(TIMESTAMP)*: microsecondeniveau

Een periode wordt uitgedrukt als een halfopen tijdsinterval *[S→E[*.

Als voorbeeld neem ik de TSQL-versie van Tabel 12. In TSQL worden de attributen *Start* en *Einde* niet meer expliciet opgenomen in de tabeldefinitie[22] maar impliciet opgenomen als een attribuut *Geldigheidsduur*. De tabel wordt als volgt gedefinieerd:

```
CREATE TABLE LotPerStal
(Boerderij INTEGER,
 Lot INTEGER,
 Stal INTEGER,
 Aantal INTEGER) AS VALIDTIME PERIOD(DATE)
```

**SQL-fragment 18:** creatie van een geldigheidsduurtabel in TSQL.

De clausule `AS VALIDTIME PERIOD(DATE)` maakt van de tabel een geldigheidsduur-tabel die als volgt kan ingevuld worden:

| Boerderij | Lot | Stal | Aantal | Geldigheidsduur |
|---|---|---|---|---|
| 1 | 137 | 1 | 17 | [07/02/1998→18/02/1998[ |
| 1 | 219 | 1 | 43 | [25/02/1998→28/02/1998[ |
| 1 | 219 | 1 | 20 | [01/03/1998→13/03/1998[ |
| 1 | 219 | 2 | 23 | [01/03/1998→13/03/1998[ |
| 1 | 219 | 2 | 43 | [14/03/1998→31/12/9999[ |
| 1 | 374 | 1 | 14 | [20/02/1998→31/12/9999[ |
| … | … | … | … | … |

**Tabel 13:** TSQL-versie van de tabel *LotPerStal*.

Een actueel tuppel kan nog steeds aan *LotPerStal* worden toegevoegd met volgende (T)SQL-opdracht:

```
INSERT INTO LotPerStal
VALUES(1,543,2,12);
```

[22] De attributen mogen nog wel expliciet gedefinieerd worden maar ze worden niet gebruikt door de temporele constructies van TSQL.

Om een tuppel met een bepaalde geldigheidsduur toe te voegen, gebruikt men volgende TSQL-opdracht:

```
VALIDTIME PERIOD '[01-03-1998 - 22-04-1998)'
INSERT INTO LotPerStal
VALUES (1,543,2,12)
```

Door de opwaartse compatibiliteit is actuele bevraging in TSQL zeer eenvoudig. De bevraging is namelijk identiek aan de SQL-bevraging van de actuele tabel:

```
Hoeveel koeien van lot 219 staan er in elke stal?
SELECT Stal, Aantal
FROM LotPerStal
WHERE Boerderij=1 AND Lot=219
```

Het DBMS selecteert automatisch alle tuppels waarvoor de *WHERE*-clausule geldt en waarvoor de geldigheidsduur de systeemtijd bevat.

Intervalbevraging gebeurt door het woord *VALIDTIME* toe te voegen aan de bevraging; hierdoor wordt aan het DBMS de opdracht gegeven alle tuppels uit de tabel te evalueren.

```
Geef een historiek van de stallen waarin ooit koeien uit lot 219 stonden.
VALIDTIME SELECT Stal, Aantal
FROM LotPerStal
WHERE Boerderij=1 AND Lot=219
```

Tot nog toe is het verschil tussen SQL en TSQL blijkbaar enkel een inkorting van de *WHERE*-clausule maar bij een intervalbevraging waarin een *join* voorkomt, wordt de expressieve kracht van TSQL pas echt duidelijk.

De intervalbevraging «*Wanneer bevonden zich verschillende loten in dezelfde stal?*» in standaard-SQL werd volledig uitgewerkt in paragraaf 4.7 als de unie van vier bevragingen. In TSQL gebeurt dezelfde bevraging als volgt:

```
VALIDTIME SELECT A.Lot, B.Lot, A.Stal
FROM LotPerStal A, LotPerStal B
WHERE A.Lot<B.Lot
  AND A.Stal=B.Stal
```

Dit is de actuele SQL-bevraging waaraan het woord *VALIDTIME* is toegevoegd. Het DBMS koppelt nu automatisch alle tuppels waarvan de geldigheidsduur een tijdstip gemeenschappelijk heeft en selecteert daaruit alle tuppels waarvoor de *WHERE*-clausule voldaan is.
Bij een waardebevraging worden de woorden *NONSEQUENCED VALIDTIME* toegevoegd.

Vb: *Geef een overzicht van de verhogingen van het aantal koeien van een bepaald lot in een stal?*

```
NONSEQUENCED VALIDTIME SELECT A.Lot, A.Aantal, B.Aantal
FROM LotPerStal A, LotPerStal B
WHERE A.Lot=B.Lot
  AND A.Stal=B.Stal
  AND A.Aantal<B.Aantal
  AND VALIDTIME(A) MEETS VALIDTIME(B)
```

De *MEETS*-comparator controleert of de geldigheidsduur van entiteit *B* onmiddellijk volgt op de geldigheidsduur van entiteit *A*.

Tot nog toe wordt TSQL nog niet ondersteund door commerciële (O)RDMBS'en. Toch zou het bijzonder praktisch zijn wanneer de databank van het SCOB gebruik zou kunnen maken van een temporele uitgebreide SQL. In [TORP97] wordt een temporeel DBMS opgebouwd aan de hand van een RDBMS. Dit gebeurt via een abstracte laag bovenop het RDBMS die alle temporele opdrachten en bevragingen vertaalt naar SQL-92. Het uitwerken van de concepten uit [TORP97] valt eigenlijk wel binnen het kader van deze thesis maar is weggelaten omwille van de uitgebreidheid van het onderwerp.

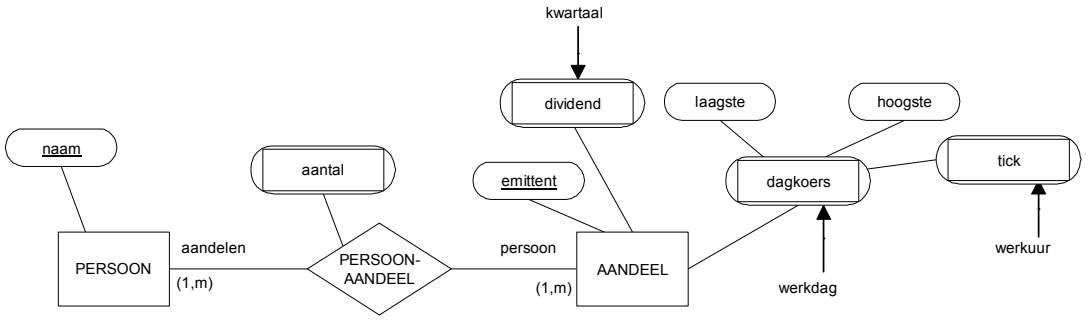
## *4.9 De vraagtaal bij ITDM*

Zoals beschreven in paragraaf 2.2.4 heeft het ITDM-model een eigen vraagtaal. Deze vraagtaal is ontwikkeld voor een gespecialiseerde bevraging van tijdreeksen, dit in tegenstelling met temporele SQL waarbij de temporele constructies geen specifieke tijdreeks-clausules bevatten.
Onderstaande bespreking van de ITDM-vraagtaal is grotendeels afkomstig uit [LEE98].

### 4.9.1 Pad-expressies

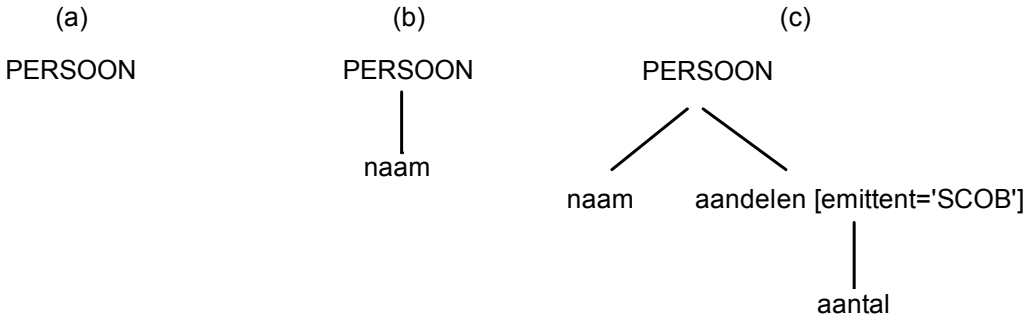
Een pad-expressie wordt gebruikt om via relatierolnamen door entiteiten te navigeren en om *join*-condities te specifiëren.

Een pad-expressie is een boomstructuur:
- de wortel is een entiteitstype
- als oudernode *o* een kindnode *k* heeft dan:
  - als *o* een entiteitstype is, dan is *k* een attribuut van *o* of een rolnaam van *o*
  - als *o* een samengesteld attribuut is, dan is *k* een samenstellend attribuut
  - als *o* een rolnaam is van entiteitstype $E_1$ dat participeert in relatie *R* met entiteitstype $E_2$, dan is *k* een attribuut van *R*, een attribuut van $E_2$ of een rolnaam van $E_2$
- aan rolnamen kan een predikaat verbonden worden




**Figuur 26:** ITDM-diagramma.

Figuur 26 is overgenomen uit paragraaf 2.2.4. Voorbeelden van pad-expressies uit deze figuur zijn:




**Figuur 27:** voorbeelden van pad-expressies.

Pad-expressies hebben ook een tekstuele voorstelling en een interpretatie:
(a) PERSOON: alle personen
(b) PERSOON.naam: naam van alle personen
(c) PERSOON.<naam,aandelen[emittent='SCOB'].aantal>: voor elke persoon, de naam en het aantal SCOB-aandelen

### 4.9.2 Niet-temporele bevraging

Een niet-temporele ITDM-bevraging heeft volgende vorm:

```
GET p1,p2, …
FROM E1 e1, E2 e2
WHERE pr
```

met $p_i$ een pad-expressie
$E_i$ een entiteitstype
$e_i$ een variabele over alle instanties van entiteitstype $E_i$
*pr* een predikaat

De bevraging wordt als volgt uitgevoerd:

- Genereer het carthesisch product tussen alle entiteitstypes uit de *FROM*-clausule.
- Evalueer het predikaat *pr* voor elk element uit het carthesisch product. Als het predikaat geldt dan wordt het element geselecteerd.
- Voor alle geselecteerde elementen wordt dan de informatie uit de *GET*-clausule getoond.

Bv. *Som van alle aandelen die Jan Janssens op dit moment bezit, de emittent en het aantal op.*

```
GET p.aandelen.<emittent, aantal>
FROM PERSOON p
WHERE naam='Jan Janssens'
```

Dezelfde bevraging zou er in SQL als volgt uitzien:

```
SELECT A.emittent, PA.aantal
FROM PersoonAandeel PA, Aandeel A, Persoon P
WHERE A.Sleutel=PA.AandeelSleutel
  AND P.Sleutel=PA.PersoonSleutel
  AND P.Naam='Jan Jansens'
```

Hierbij wordt de kracht van pad-expressies getoond. In het SQL-fragment stellen twee regels uit de *WHERE*-clausule de *join*-conditie. In het ITDM-fragment is deze conditie veel transparanter in de pad-expressie. Ook de *FROM*-clausule in ITDM is korter dan de *FROM*-clausule uit het SQL-fragment. Toch is de *FROM*-clausule in de ITDM-bevraging niet altijd even duidelijk. In dit voorbeeld worden blijkbaar gegevens opgevraagd uit de tabel *Persoon*, terwijl het resultaat enkel gegevens uit de tabel *Aandeel* toont. Dit is dan weer een nadeel van pad-expressies maar het geeft tegelijk wel weer hoe nauw een ITDM-bevraging aanleunt bij een ITDM-diagramma. Zonder het diagramma kan een bevraging zeer verwarrend overkomen.

### 4.9.3 Temporele bewerkingen

Bij *temporele projectie* in ITDM wordt het getoonde resultaat beperkt tot een bepaald tijdsinterval. Dit wordt verwezenlijkt door bij de pad-expressie het gewenste tijdsinterval te plaatsen.

Bv. *Geef de koers van alle aandelen op die Jan Janssens tijdens het jaar 1998 bezat.*

```
GET p.aandelen.dagkoers.hoogste:[01/01/1998,31/12/1998]
FROM PERSOON p
WHERE naam='Jan Janssens'
```

Een *predikaat* over een entiteit met temporele attributen is een functie met als domein de levensduur van de entiteit en als beeld de verzameling *{waar,onwaar}*. De geldigheidsduur van dergelijk predikaat is de verzameling intervallen tijdens dewelke het predikaat geldt (of nog: de functiewaarde *waar* is), deze geldigheidsduur wordt genoteerd als *[[ pr(e) ]]*. De formele definitie van *[[ pr(e) ]]* is $pr^{-1}(waar)$.

Bv. Stel dat het aandeel uitgegeven door SCOB volgende tijdreeks voor het attribuut *dividend* heeft:

```
{[01/01/1998,31/03/1998]→495,
 [01/04/1998,30/06/1998]→505,
 [01/07/1998,30/09/1998]→508,
 [01/10/1998,31/12/1998]→496,
 [01/01/1999,31/03/1999]→501,
 [01/04/1999,heden]→500}
```

Wanneer *pr* wordt gedefinieerd als *dividend kleiner dan 500* dan is *pr(SCOB-aandeel)* gelijk aan

```
{[01/01/1998,31/03/1998]→waar,
 [01/04/1998,30/09/1998]→onwaar,
 [01/10/1998,31/12/1998]→waar,
 [01/01/1999,heden]→onwaar}
```

Merk op dat dit resultaat intervalgemaximaliseerd is.
*[[ pr(SCOB-aandeel) ]]* is bijgevolg *{[01/01/1998,31/03/1998], [01/10/1998,31/12/1998]}.*

Een *temporeel selectiepredikaat* is een booleaanse expressie die twee temporele elementen vergelijkt d.m.v de verzamelingoperatoren uit {∪,∩,⊃,⊇,⊄,⊂,⊆,∩,≠,=} waarbij tenminste één van de operanden de geldigheidsduur van een predikaat is.

Vb. *Geef de personen die gedurende het hele eerste kwartaal van 1998 een SCOB-aandeel bezaten.*

```
GET p.naam
FROM PERSOON
WHERE [[ p.aandelen.emittent='SCOB' ]] ⊇ [1/1/1998, 31/3/1998]
```

*Geef de personen die ooit gedurende het eerste kwartaal van 1998 een SCOB-aandeel bezaten.*

```
GET p.naam
FROM PERSOON
WHERE ([[ p.aandelen.emittent='SCOB' ]] ∩ [1/1/1998, 31/3/1998])!=NULL
```

### 4.9.4 Tijdselectiefuncties

Bij sommige temporele bevragingen moet het resultaat van de bevraging niet de attribuutwaarde zijn maar wel de geldigheidsduur van die attribuutwaarde.
Hiervoor zijn twee tijdselectiefuncties gedefinieerd:

- *I_SELECT(i,T)*: retourneert het *i*-de interval uit de tijdreeks *T* waarbij *i* een natuurlijk getal is of de constante *FIRST* of *LAST*.
  Voor de tijdreeks *dividend* uit de vorige paragraaf geldt:
  *I_SELECT(3,dividend) = [01/07/1998,30/09/1998]*
- *T_SELECT(i,I)*: retourneert het *i*-de tijdselement uit het interval *I*.
  *T_SELECT(3,[01/01/1998,31/03/1998]) = 03/01/1998*

Wanneer een tijdselectiefunctie wordt gebruikt in een ITDM-bevraging dan moet de *GET TIME*-clausule gebruikt worden.

Bv. *Wanneer had het SCOB-aandeel een eerste dividend?*

```
GET TIME T_SELECT(FIRST, I_SELECT(FIRST,a.dividend))
FROM AANDEEL a
WHERE emittent='SCOB'
```

### 4.9.5 Temporele aggregaties

De gekende aggregatiefuncties *SUM*, *AVERAGE*, *COUNT*, *MIN*, *MAX*,… worden geherdefinieerd in een temporele versie; deze worden aangeduid met de beginletter T.

Bv. *Wat is het gemiddelde dividend van het SCOB-aandeel in 1998?*

```
GET TAVERAGE(a.dividend):[1/1/1998, 31/12/1998]
FROM AANDEEL a
WHERE a.emittent='SCOB'
```

Voor de aggregatiefuncties *TMAX* en *TMIN* wordt een geldigheidsduur gedefinieerd; deze wordt genoteerd als *[[f(a): [ S→E] ]]* met *f∈{TMAX, TMIN}*.

Bv. *Wanneer behaalde het SCOB-aandeel de hoogste koers in 1998?*

```
GET TIME [[TMAX(a.dagkoers.hoogste):[01/01/1998,31/12/1998] ]]
FROM AANDEEL a
WHERE a.emittent='SCOB'
```

### 4.9.6 Tijdreeksvensters

Tot nog toe zijn enkel de algemene temporele aspecten van de ITDM-vraagtaal besproken. ITDM heeft echter een belangrijke constructie voor tijdreeksbevragingen, nl. tijdreeksvensters. Een tijdreeksvenster kan een waarde of een reeks waarden selecteren uit een bepaalde tijdreeks.

Er zijn vijf types:

- *[$t_1$,$t_2$]*: alle waarden tussen de tijdstippen $t_1$ en $t_2$ inclusief de waarden op $t_1$ en $t_2$
- *[t,%i%]*: *i* opeenvolgende waarden vanaf tijdstip *t*. Als er geen waarde bestaat op tijdstip *t* dan wordt de eerste daaropvolgende waarde genomen.
- *[t;%i%]*: analoog met het vorige type, alleen wordt hier enkel de *i*-de waarde geselecteerd
- *[t,%iT%]*: alle waarden tussen de tijdstippen *t* en *t+iG, G* is een tijdseenheid zoals *DAY* of *YEAR*
- *[t;%iT%]*: de waarde op tijdstip *t+iG*

De notatie van een tijdreeksvenster wordt geïllustreerd door volgend voorbeeld:

*Geef de eerste vijf koersen van het SCOB-aandeel in 1998.*

```
GET a.dagkoers.hoogste: [01/01/1998, %5%]
FROM AANDEEL a
WHERE a.emittent='SCOB'
```

Een complexer voorbeeld is:

*Geef de vijf koersen die volgen op de hoogste koers van het SCOB-aandeel in 1998.*

```
GET a.dagkoers.hoogste:
      [ [[TMAX(a.dagkoers.hoogste):[1/1/1998,31/12/1998] ]]: %5%]
FROM AANDEEL a
WHERE a.emittent='SCOB'
```

Merk op dat hoewel het maximum geselecteerd wordt uit de koersen van 1998, de daaropvolgende koersen niet noodzakelijk uit 1998 geselecteerd worden. Wanneer de hoogste koers valt op 31/12/1998, dan vallen de vijf daaropvolgende koersen in 1999.

Tijdreeksvensters zijn statisch. Voor bevragingen als «*Geef voor elk kwartaal van 1998 de hoogste koers van het SCOB-aandeel.*» maakt ITDM gebruik van schuivende tijdreeksvensters waarop telkens een aggregatiefunctie wordt uitgevoerd. Schuivende tijdreeksvensters worden gedefinieerd aan de hand van een tijdreeksvenster van het type *[$t_1$,$t_2$]* en twee tijdsduren van de vorm *%i%*, *%iT%* of *%calendar%*; *calendar* specifieert een vooraf gekend patroon zoals *kwartaal*, *semester*, *maand*…

Een venster met een breedte van tien dagen dat met stappen van vijf dagen schuift over het jaar 1998 wordt als volgt beschreven:

```
[01/01/1998, 31/12/1998] FOR %10DAY% INCREMENT %5DAY%
```

De *FOR*-clausule specifieert de breedte van het venster, de *INCREMENT*-clausule specifieert de grootte van de beweging van het venster.

Een voorbeeld van een tijdreeksbevraging met een schuivend venster: *Geef een tiendaags gemiddelde van de dagelijkse hoogste koers van het SCOB-aandeel, telkens met een sprong van 5 dagen.*

```
GET TAVERAGE(a.dagkoers.hoogste)
      : [01/01/1998,31/12/1998] FOR %10DAY% INCREMENT %5DAY%
FROM AANDEEL a
WHERE a.emittent='SCOB'
```

De mogelijkheden van een vraagtaal zoals deze zijn groot, zeker in de SCOB-databank. In hoofdstuk 7 wordt besproken hoe een dergelijke vraagtaal kan worden ingepast in het SCOB-project.

Hiermee eindigt dit hoofdstuk en meteen ook het theoretische gedeelte van dit thesisdocument. De hoofdstukken die volgen zijn toegelegd op de praktische kant van het SCOB-project.

# **Hoofdstuk 5 :** Modellering van de categorie-indeling

De meeste elementen uit de SCOB-miniwereld kunnen vrij eenvoudig gemodelleerd worden, al dan niet gebruik makend van tijdreeksen en geldigheidsduur-tabellen. Een grote uitzondering hierop is het noteringssysteem voor aandelen en obligaties op een beurs. Het probleem is dat dit systeem overeenkomt met een boomstructuur die in de loop van de tijd verschillende soorten veranderingen kan ondergaan. Het relationele model is niet erg geschikt om boomstructuren te stockeren, laat staan gedurig veranderende boomstructuren. Dit hoofdstuk omvat daarom een stap-voor-stap modellering van het eigenlijke koersboek dat gebruikt wordt door het SCOB tot een relationeel schema. Bij elke stap worden bepaalde keuzes gemaakt en verantwoord.

## *5.1 Het koersboek*

In een koersboek[23] worden dagelijks alle koersen van de op een beurs verhandelde effecten genoteerd. De effecten worden gegroepeerd in categorieën die de beursstructuur weerspiegelen. In de vroegste koersboeken uit het SCOB-archief is deze structuur bijzonder eenvoudig.

> Bv. In 1832 zijn er twee categorieën: «*Publieke Fondsen*» en «*Aandelen & Obligaties*».

In de loop der tijden is de structuur complexer geworden, zowel qua aantal als qua structuur van de categorieën.

> Bv. In 1856 wordt de categorie «*Aandelen & Obligaties*» onderverdeeld in 6 subcategorieën: «*Banken en Spaarkassen*», «*Spoorwegen*», «*Zinkfabrieken*», «*Stof*», «*Linnen*», «*Kolen & Hoog-ovens*» en «*Diversen*».
> In 1878 zijn er reeds 11 categorieën en 9 subcategorieën.

In de tweede helft van de 20ste eeuw komt het geregeld voor dat subcategorieën zelf ook nog eens worden onderverdeeld. Voor zover geweten komt dit niet voor in de periode 1832-1950.

> Bv. In 1962 is de categorie «*Aandelen*» onderverdeeld in 23 subcategorieën, waarvan er 4 verder zijn onderverdeeld. Zo is de subcategorie «*Diverse Industrieën*» verder onderverdeeld in de sub-subcategorieën «*Papierwaren*» en «*Petroleum*».

De subcategorieën worden niet enkel gebruikt om bv. een categorie «*Aandelen*» op te splitsen in industrieën maar ook om de buitenlandse subcategorieën op te nemen.

> Bv. Een vanzelfsprekend voorbeeld dat sinds 1832[24]-1999 geldt, is de categorie «*Buitenlandse aandelen*» waarin een subcategorie voorkomt voor elk land waarvan effecten genoteerd staan.

Wanneer de categorieën waarin buitenlandse effecten per land zijn verdeeld buiten beschouwing worden gelaten, dan blijkt uit een analyse[25] van dr. Buelens dat de koersboekstructuur in de periode 1832-1980 ongeveer 90 keer wordt aangepast; soms zijn deze aanpassingen zeer drastisch waarbij een geheel nieuwe structuur wordt ingevoerd, soms handelt het slechts om een naamsverandering van een categorie of het wegvallen van een subcategorie. Al bij al heeft de koersboekstructuur dus een redelijk variërende vorm.

[23] Appendix C toont een uitreksel uit een koersboek.
[24] In 1832 is er nog geen categorie «*Buitenlandse aandelen*» maar de categorie «*Aandelen & obligaties*» is onderverdeeld in alle landen waarvan aandelen genoteerd zijn; «*België*» is dus één subcategorie.
[25] Een overzicht van alle structuurveranderingen in de koersboeken van de beurs van Brussel in de periode 1832-1980 is te vinden in [BUEL99].

## *5.2 Van koersboek naar datastructuur*

De bedoeling van deze paragraaf is om een datastructuur uit te werken die zo nauwkeurig mogelijk overeenstemt met de koersboekstructuur: de onderverdeling in categorieën en subcategorieën is daarbij belangrijk maar ook de volgorde van de categorieën en van de effecten binnen een categorie. Verder mag men niet uit het oog verliezen dat deze datastructuur zal worden opgeslagen in een relationele databank.

Een voor de hand liggende keuze is een boomstructuur met beperkte diepte en een variabel aantal kinderen per node:

- de wortelnode is de beurs
- alle nodes met diepte 1 zijn de categorieën
- de nodes met een diepte groter dan 1 zijn ofwel effecten die genoteerd zijn op de beurscategorie uit de oudernode ofwel subcategorieën van de oudernode
- de bladnodes zijn de effectnoteringen

Stel dat de Beurs van Brussel volgende koersboekstructuur heeft:

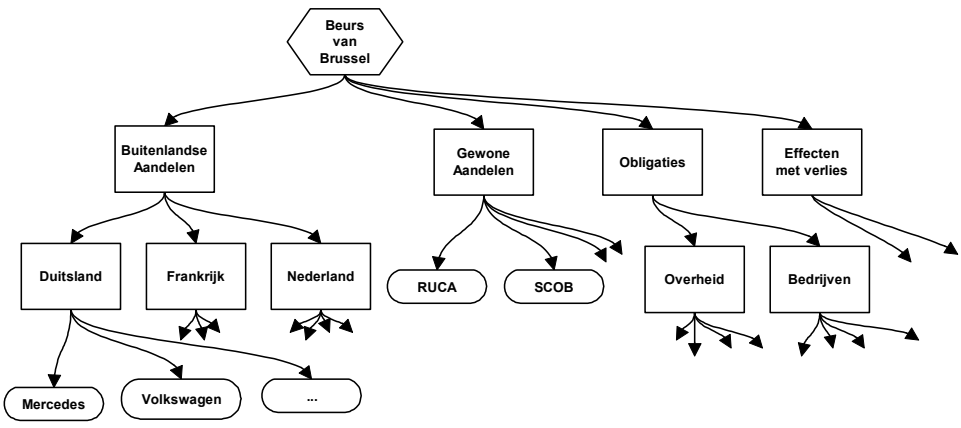
1.Buitenlandse aandelen
- A.Duitsland
  - *Mercedes*
  - *Volkswagen*
  - …
- B.Frankrijk
  - *Citroën*
  - …
- C.Nederland
  - *Volvo*
  - …

2.Gewone aandelen
- *RUCA*
- *SCOB*
- *UA*
- *UFSIA*
- *UIA*
- …

3.Obligaties
- A.Overheid
  - *Provincie Antwerpen*
  - *Brussels Gewest*
  - …
- B.Bedrijven
  - *Gemeentekrediet*
  - *KBC*
  - …

4.Effecten met verlies
- *Generale Maatschappij*
- …

Dan heeft de bijbehorende datastructuur de vorm van Figuur 28. Bij sommige pijlen in de figuur is de doelnode weggelaten, dit is bij wijze van afkorting gebeurd, eigenlijk wijzen al die pijlen naar een effect.




**Figuur 28:** directe boomvoorstelling van de koersboekstructuur.

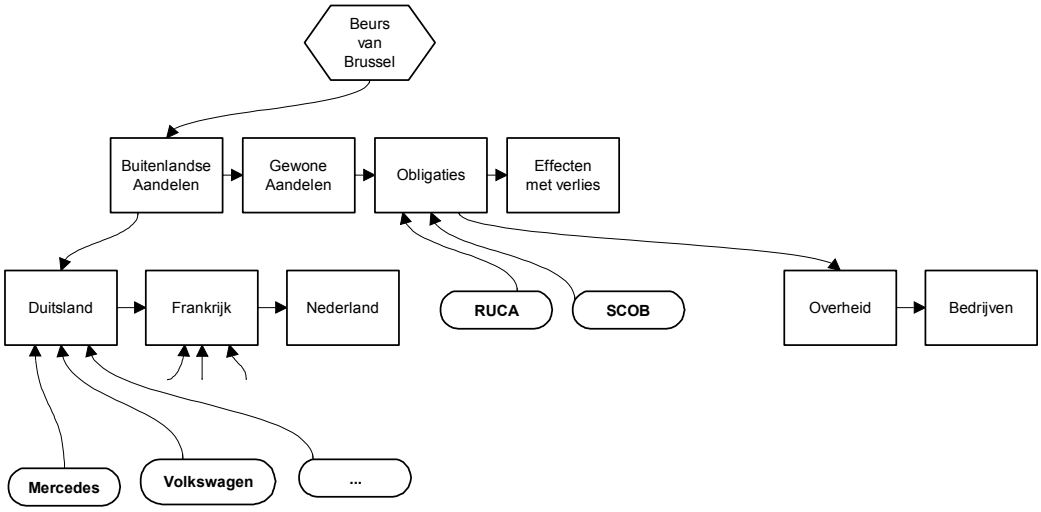
Het gebruik van dergelijke boomstructuur schept de nodige problemen:

- De nodes hebben geen uniform type: wanneer we veronderstellen dat categorieën en subcategorieën van hetzelfde type zijn, dan is er sprake van 3 nodetypes, namelijk *Beurs*, *Categorie* en *Effect*.
- De boom is niet typegelaagd d.w.z. dat er lagen zijn in de boom waarin verschillende nodetypes voorkomen. Op diepte 2 in Figuur 28 komt zowel het type *Categorie* («*Duitsland*», «*Frankrijk*», «*Nederland*», «*Overheid*», …) als het type *Effect* («*RUCA*»,«*SCOB*», …) voor.
- Het aantal kinderen is variabel: zo kan in realiteit een categorie een twintigtal subcategorieën omvatten en kunnen op een (sub-)categorie honderden effecten genoteerd staan.
- Deze boom heeft een mooie statische structuur maar veranderingen aan de boom zijn niet vanzelfsprekend. Stel dat in Figuur 28 tussen de categorie «*Gewone aandelen*» en «*Obligaties*» een nieuwe categorie «*Buitengewone aandelen*» moet worden toegevoegd. Vermits volgorde belangrijk is in de boom, moet de pijl naar de nieuwe subcategorie geplaatst worden tussen de twee al bestaande pijlen. Wanneer de kindpijlen van een node worden voorgesteld door een rij (*array*) dan moet een gedeelte van de rij worden opgeschoven.

Vooral de laatste twee puntjes zijn niet flexibel op te lossen in het relationele model, daarom is de boomstructuur abstracter gemaakt:

- De pijlen naar bladnodes worden omgekeerd, de bladnodes wijzen nu dus naar interne nodes of nog: de effecten wijzen naar de categorie waarin ze genoteerd staan. Effecten worden bijgevolg niet langer rechtstreeks opgenomen in de boomstructuur.
- Elke interne node (inclusief wortel) heeft maximaal 1 kindnode, namelijk een pijl naar de eerste kindnode uit de vorige boomstructuur.
- Elke interne node heeft een extra pijl naar de volgende kindnode van de oudernode.

Volgens deze nieuwe boomstructuur wordt Figuur 28 omgevormd tot Figuur 29.

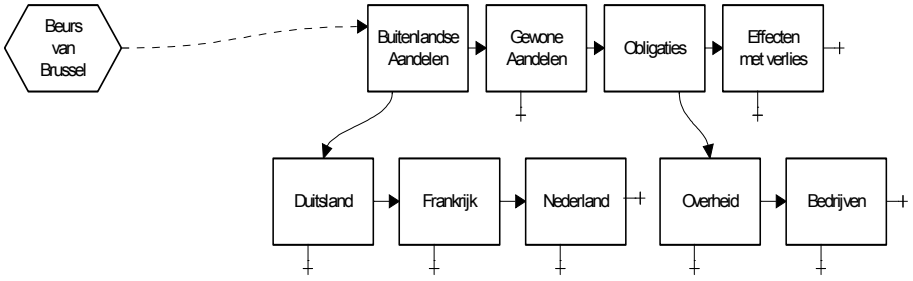



**Figuur 29:** een meer realistische boomstructuur voor een koersboek.

Deze nieuwe structuur heeft een paar opmerkelijke voordelen t.o.v. de vorige:

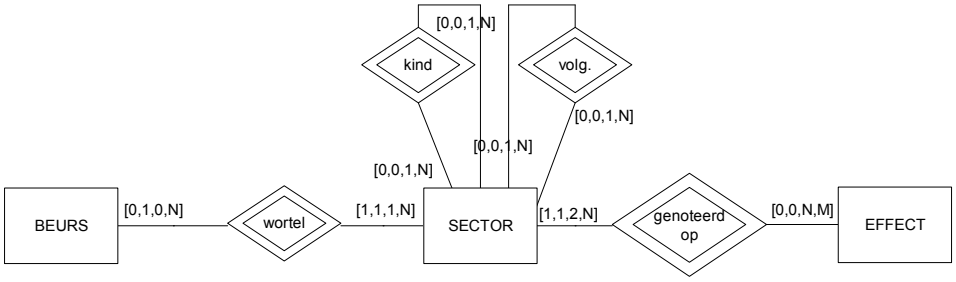
- Door de pijlen van C*ategorie* naar *Effect* om te keren maken effecten geen deel meer uit van de boomstructuur. Hierdoor wordt de grootte van de boom drastisch ingesnoeid. Tevens wordt een generalisatieprobleem opgelost: in een beurscategorie kunnen namelijk aandelen en obligaties tezamen vermeld staan. Zowel het type *Aandeel* als het type *Obligatie* zal dus een pijl naar een *Categorie* bevatten terwijl in omgekeerde richting een *Categorie* een massa *Aandeel*- en/of *Obligatie*-pijlen zou bevatten.
- Het invoegen van een subcategorie gebeurt nu door de klassieke bewerkingen van lineaire lijsten.
- Er is geen variabel aantal pijlen meer per boomnode. Een *Beurs* heeft steeds één *EersteCategorie*-pijl, namelijk voor de eerste categorie die voorkomt in het koersboek. Een *Categorie* heeft steeds één *VolgendeCategorie*-pijl, tenzij het gaat om de laatste categorie uit de lijst; in dit geval wordt de wijzer op *NULL* gezet hetgeen ondersteund wordt door het relationeel model. Hetzelfde geldt voor *EersteSubcategorie*-pijlen: ofwel heeft een *Categorie* één *EersteSubcategorie*, ofwel geen (*NULL*-waarde).

Er is nu nog één probleem: de boom bevat nog steeds twee types nodes, de wortel is namelijk van het type *Beurs* terwijl alle andere nodes (ook de subcategorieën) van het type *Categorie* zijn. Dit wordt opgelost door de wortelnode los te koppelen van de structuur. Wat overblijft zijn enkel nog *Categorie*-nodes, zie Figuur 30.




**Figuur 30:** gestripte versie van de abstracte boomstructuur**.**

In het ACER-model kan deze gestripte boomstructuur als volgt gemodelleerd worden; voor de eenvoud zijn de attributen weggelaten:




**Figuur 31:** ACER-model voor de koersboekstructuur.

Alle pijlen in de boomstructuur worden voorgesteld door temporele relaties. Hier eindigt de eigenlijke modellering van de koersboekstructuur. Het gepresenteerde model moet nu omgezet worden naar een relationeel schema.

## *5.3 Van structuur naar relationeel schema*

In deze paragraaf wordt het ontwikkelde model vertaald naar SQL, hierbij ligt de klemtoon op de tabelattributen die bijdragen tot de opslag van de structuur, andere attributen worden buiten beschouwing gelaten. Tevens wordt het temporele aspect niet in rekening gebracht.
Er is sprake van drie types (*Beurs* , *Categorie* en *Aandeel*) en vier relaties (*wortel*, *kind*, *volgend* en *genoteerdop*). Dit alles wordt geïmplementeerd in volgende drie tabellen:

```
CREATE TABLE Beurs
( BeursSleutel INTEGER PRIMARY KEY,
  BeursNaam VARCHAR(25),
  EersteCategorie INTEGER REFERENCES Categorie(CategorieSleutel),
  ... )

CREATE TABLE Categorie
( CategorieSleutel INTEGER PRIMARY KEY,
  CategorieNaam VARCHAR(25),
  EersteSubCategorie INTEGER REFERENCES Categorie(CategorieSleutel),
  VolgendeCategorie  LONG REFERENCES Categorie(CategorieSleutel),
  ... )

CREATE TABLE Aandeel
( AandeelSleutel INTEGER PRIMARY KEY
  AandeelNaam VARCHAR(25),
  GenoteerdOp INTEGER REFERENCES Categorie(CategorieSleutel),
  ... )
```

**SQL-fragment 19:** creatie van actuele structuurtabellen.

Zelfs in de actuele versie is het onderhoud van de tabellen geen vanzelfsprekendheid want het model maakt verwarrende situaties mogelijk: stel dat op dezelfde dag de categorie «*Hoogovens*» wordt verwijderd en de categorie «*Banken & Verzekeringen*» wordt gesplitst in «*Banken*» en «*Verzekeringen*». Deze veranderingen kunnen op verschillende manieren worden verwezenlijkt:

- een vrij logische manier: er worden twee nieuwe categorieën aangemaakt.
  - verwijder «*Hoogovens*» en «*Banken & Verzekeringen*» uit de structuur
  - voeg «*Banken*» en «*Verzekeringen*» toe
  - verplaats de onthoofde effecten uit «*Banken & Verzekeringen*» naar «*Banken*» of «*Verzekeringen*»

- een ruimtebesparende manier: «*Banken & Verzekeringen*» en «*Hoogovens*» worden hergebruikt.
  - hernoem «*Banken & Verzekeringen*» in «*Banken*»
  - hernoem «*Hoogovens*» in «*Verzekeringen*»
  - verplaats de verzekeringsaandelen van «*Banken*» naar «*Verzekeringen*»

- een minder vanzelfsprekende maar toch correcte manier:
  - hernoem «*Hoogovens*» in «*Verzekeringen*»
  - verwijder «*Banken & Verzekeringen*»
  - voeg «*Banken*» toe
  - verplaats de onthoofde aandelen naar «*Banken*» of «*Verzekeringen*»

Er zijn nog andere mogelijkheden waarbij het uiteindelijke resultaat hetzelfde is: een correcte weergave van de koersboekstructuur en de genoteerde aandelen.

Volgens financieel-economen moet ernaar gestreefd worden om zo weinig mogelijk *Categorie*-instanties in de databank te stockeren en wel om deze reden: één van de eerste analyses die door hen zal worden uitgevoerd is de vraag: «*Hoe evolueren de aandelen binnen een bepaalde categorie?*»[26] Telkens wanneer de categoriestructuur wijzigt, worden meestal ook heel wat *GenoteerdOp*-attribuutwaarden in de tabel *Aandeel* aangepast en hier wringt het schoentje: als een aandeel van categorie verandert, moet handmatig gecontroleerd of de nieuwe categorie nog steeds van het beoogde type is, of nog: of het aandeel nog in aanmerking komt voor de huidige analyse. Het invullen van de *Categorie*-tabel moet bijgevolg met de nodige omzichtigheid gebeuren. Er is echter geen exacte methodiek te geven, er blijven randgevallen waarbij de keuze tussen het hergebruik van een categorie of het invoeren van een nieuwe categorie vaag blijft.

Ondertussen is aan de hand van [BUEL99] een categoriestructuur in de SCOB-databank ingegeven met geldigheidsduur-tabellen dus zonder tijdreeksen. Deze structuur zal waarschijnlijk gehandhaafd blijven in Oracle8.

Het oorspronkelijke doel van bovenstaande modellering is bereikt: een relationeel schema waarin de koersboekstructuur gestockeerd kan worden. Toch moet nog één essentieel aspect van de modellering bekeken worden: hoe wordt de oorspronkelijke koersboekstructuur gereconstrueerd uit het relationeel schema? Dat wordt beschreven is de volgende paragraaf.

## *5.4 Reconstructie van de structuurboom*

Om de structuurboom te reconstrueren moeten twee parameters gekend zijn: *beurs* en *datum*. Met behulp van deze twee parameters moeten aan de databank de juiste bevragingen gesteld worden. Voor de eenvoud bespreek ik hier de niet-temporele bevragingen dus de parameter *datum* is overbodig.
Er is ook gekozen om in de Java-code de boomstructuur te stockeren via de klasse *JTree()*[27]; dit is niet onbelangrijk om weten vermits deze boomstructuur van blad naar wortel moet worden opgebouwd. In de pseudocode wordt de naam *JTree* gemakshalve vertaald naar *BoomNode*.

[26] Tijdens het academiejaar 1999-2000 zullen een viertal TEW-thesisstudenten zich met deze vraag bezighouden.
[27] *JTree()* maakt deel uit van de *Swing*-standaardklassen uit Java 1.2. Lees de Java-documentatie voor meer informatie.

De reconstructie van de boomstructuur gebeurt als volgt:

- De module die de categorieboom nodig heeft, roept de functie *CreeerBoom(beurs)* op. Deze functie zoekt op wat de wortelnode is van de boomstructuur(lijn 3-7). Zoals verklaard in de vorige paragraaf is deze informatie terug te vinden in de tabel *Beurs*.
- De functie *MaakBoom(categorie, boomnode)* doorloopt de tabel *Categorie* en verricht dus de eigenlijke reconstructie van de boomstructuur. Deze functie zoekt alle *sibling*-nodes[28] van de huidige categorie. Lijnen 15-24 bevragen de tabel *Categorie* zodat alle informatie over de huidige categorie volledig bekend wordt, het betreft hier voornamelijk de attributen *VolgendeCategorie* en *EersteSubCategorie*. Deze informatie wordt dan geïncapsuleerd[29] in het ADT *DBCategorie* (lijnen 27-30).
  Lijn 33: Indien de categorie subcategorieën heeft, wordt *MaakBoom* recursief opgeroepen, waardoor eerst deze subboom wordt geconstrueerd; dit is een gevolg van het feit dat de Java-klasse *JTree()* van blad naar wortel moet worden opgebouwd. In het andere geval (lijn 34-36) is de huidige categorie een bladnode en moeten de tabellen *Aandeel* en *Obligatie* doorlopen worden om alle effecten die op deze categorie genoteerd staan toe te voegen.
  Lijn 37: op dit punt is alle informatie van de huidige categorie (en de eventuele subboom) gereconstrueerd en kan de *categorie*-node worden toegevoegd aan *boomNode*, waarna het hele proces wordt herhaald voor de volgende *sibling*-node voor de huidige categorie (lijnen 38-39).

```
01 CreeerBoom(beurs):BoomNode
02 { // executeQuery voert een SQL-query uit en stockeert het resultaat in rs
03   rs=executeQuery(" SELECT EersteCategorie"+
04                   " FROM Beurs "+
05                   " WHERE BeursSleutel="+beurs;
06
07   long eersteKind=rs.getLong("EersteCategorie"); // lees de waarde uit rs
08   BoomNode wortel=new BoomNode;
09   MaakBoom(eersteKind,wortel);
10   return wortel; // Het retourneren van de wortel staat gelijk met het retourneren van de hele boom
11 }
12
13 MaakBoom(categorie,boomNode)
14 { do
15   { rs=executeQuery(" SELECT VolgendeCategorie"+
16                     " FROM Categorie"+
17                     " WHERE CategorieSleutel="+categorie);
18     volgende=rs.getLong("VolgendeCategorie"); //getal of null
19
20     rs=stmt.executeQuery(" SELECT EersteSubCategorie"+
21                          " FROM Categorie"+
22                          " WHERE CategorieSleutel="+categorie);
23
24     kind=rs.getLong("EersteSubCategorie");//getal of null
25
26     // DBCategorie is een ADT dat een Categorie-tuppel incapsuleert
27     DBCategorie huidige=new DBCategorie(categorie,volgende,kind);
28
29     // creëer een Java-boomnode die het ADT huidige bevat
30     BoomNode nieuweNode=new BoomNode(huidige);
31
32      // controleer of er recursief verder gebouwd moet worden
33     if (kind!=NULL) MaakBoom (kind,nieuweNode);
34     else
35     { //  voeg alle effecten die genoteerd staan op de categorie toe aan NieuweNode
36     }
37     boomNode.add(nieuweNode);    // voeg nieuweNode als kindnode aan boomNode
38     categorie=volgende; // schakel over naar de volgende categorie op het huidige niveau
39   } while (categorie!=NULL)
40 }
```

De temporele versie van dit algoritme is geïmplementeerd in de koersmodule. Het meest opvallende verschil met bovenstaand algoritme is dat de SQL-bevragingen temporeel zijn.

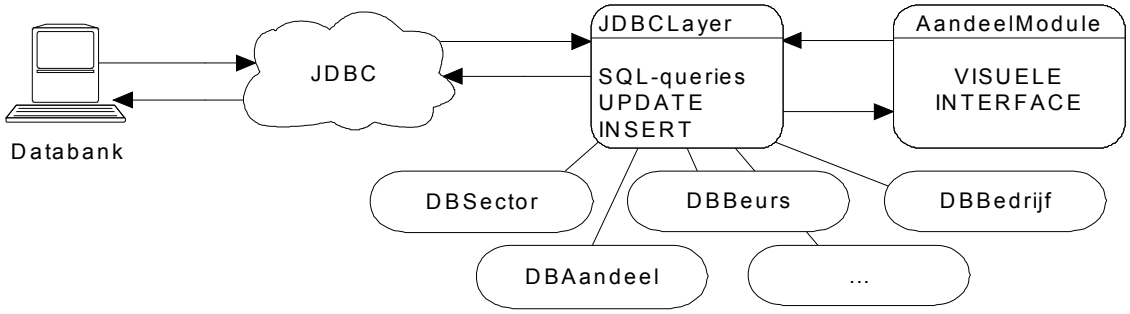
[28] *Sibling*-nodes zijn kindnodes van een zelfde oudernode. Er is geen goed Nederlands alternatief voor het woord *sibling*.
[29] In paragraaf 6.1 wordt meer uitleg gegeven bij het gebruik van ADT's in de SCOB-programmatuur.

# Hoofdstuk 6 : De koersmodule

Het praktische gedeelte van deze thesis omhelst voornamelijk de ontwikkeling van een Java-module die SCOB-medewerkers in staat stelt om koersgegevens in de databank te stockeren. Het onderhoud van het noteringssysteem en de bevraging van de bijbehorende boomstructuur maken hier een fundamenteel deel van uit. Gezien het hoge aantal in te geven koersen moet de module een invoerder in staat stellen om zo vlot mogelijk gegevens in te typen.
Het heeft weinig zin om hier een volledige bespreking van de module te geven of de code lijn per lijn te analyseren maar toch worden in dit hoofdstuk een paar belangrijke aspecten van de module besproken.

## *6.1 Ontwerp van de module*

De module is ontwikkeld in Java 1.2, hiermee is ook meteen de beslissing genomen om de communicatie met de onderliggende databank te bewerkstelligen via JDBC (*Java Database Connectivity*). Vermits deze module een eerste stap is in een groter (modulair) sofwareproject, is getracht om de klasse-hiërarchie zo te ontwerpen dat toekomstige modules eenvoudig kunnen worden gekoppeld aan de huidige module. Deze hiërarchie is voorgesteld in Figuur 32.




**Figuur 32:** klasse-hiërarchie van de module.

- De basis voor de module is uiteraard een databank, tot nog toe is hiervoor *MS-Access '97* gebruikt maar aangezien er gebruik gemaakt wordt van *JDBC* moet een overschakeling naar een *Oracle8* databank hier zonder problemen mogelijk zijn.
- Zoals gezegd gebeurt de communicatie met de databank via *JDBC*, deze klasse is in Figuur 32 voorgesteld als een wolk omdat deze klasse reeds volledig geïntegreerd zit in Java en de interne structuur verder van geen belang is.
- De klasse *JDBCLayer* is een belangrijke klasse: hierin worden alle datastructuren gedefinieerd die door de klasse *AandeelModule* (en alle toekomstige invoer- en analysemodules) gebruikt zullen worden. De meeste van deze datastructuren zijn eenvoudige ADT's die rijen uit een SQL-resultaatverzameling incapsuleren. Hun definities zijn terug te vinden als geneste klassen: *DBCategorie*, *DBAandeel*, *DBBeurs*, *DBBedrijf, …* In *JDBCLayer* wordt ook een connectie gelegd met de databank (via JDBC); deze connectie wordt constant opengehouden tot aan het beëindigen van de klasse.
  Naast datastructuren bevat de klasse ook alle functies die communicatie met de databank vereisen; het gaat hierbij om SQL-bevragingen en *update*-opdrachten. Het resultaat van de SQL-opdrachten wordt omgezet naar de vernoemde ADT's. Hierdoor wordt de communicatie met de databank volledig afgeschermd van alle bovenliggende modules waardoor alle structurele aanpassingen in de onderliggende databank enkel invloed hebben op de code van *JDBCLayer*.
- De klasse *AandeelModule* is volledig verantwoordelijk voor de visuele interface: tekenen van de interface, interactie met de gebruiker, controleren van de correctheid van invoervelden. Deze klasse voert geen rechtstreekse communicatie met de databank maar geeft *JDBCLayer* opdrachten om deze communicatie te bewerkstelligen en het resultaat te retourneren.

Voor de eigenlijke implementatie van de module is eerst nagegaan welke SQL-bevragingen nodig zijn om de *interface* van voldoende gegevens te voorzien en welke de bijbehorende ADT's zijn. De meeste van de bevragingen zijn rechttoe-rechtaan '*SELECT FROM WHERE*'-opdrachten. De grootste hindernis was het reconstrueren van de koersboekstructuur zoals besproken in paragraaf 5.4.
In een eerste fase is de *interface* horizontaal geïmplementeerd, d.w.z. dat getracht is zo snel mogelijk het uitzicht van de module vast te leggen zonder dat de *interface*-elementen (knoppen, tekstvelden, comboboxen, …) functioneel zijn. Na verschillende testsessies met dr. Beulens is dan een consensus ontstaan tussen de noden van de invoerder en de mogelijkheden van de module, zodat begin februari een eerste volledige implementatie getest kon worden. Eind februari is dan een eerste volledig functionele versie in gebruik genomen zodat effectief begonnen kon worden met het invullen van aandeel- en koersgegevens.

Naast dr. Buelens maken ook een vijftal jobstudenten gebruik van de module. In de mate van het mogelijke is ook aan hen gevraagd om de module te evalueren maar hieruit zijn geen nieuwe tekorten of vereisten voort gekomen waaruit ik denk te mogen afleiden dat de module gebruiksvriendelijk genoeg is om dienst te doen als invoermodule. Appendix E toont een schermvoorbeeld van de module.

## *6.2 Problemen met INSERT-opdrachten in Microsoft Access*

Om via JDBC in Java 1.2 een *insert*-opdracht uit te kunnen voeren op de databank hebben we een connectie nodig met de databank; in deze paragraaf wordt verondersteld dat deze connectie gelegd is en toegankelijk is via de variable *con*.
In de Access-versie van de SCOB-databank is de geldigheidsduur-tabel *AandeelKBSNaam* als volgt gedefinieerd:

```
Table AandeelKBSNaam:
      AandeelSleutel LONG
      Naam  String(255)
      Begin Date
      Einde Date
      Primary Key (AandeelSleutel, Begin)
```

Het volgende Java-fragment zou aan de tabel *AandeelKBSNaam* een nieuw tuppel moeten toevoegen. Voorafgaand aan dit fragment is gecontroleerd dat het koppel (*AandeelSleutel*, *Begin*) een nieuwe unieke sleutel vormt. Het fragment mag dus geen problemen opleveren.

```
1     stmt=con.createStatement();
2     query="INSERT INTO AandeelKBSNaam(AandeelSleutel,Naam,Begin)" +
3           " VALUES  ("+stockKey+","+
4                      "'"+newName+"',"
5                      "{d '"+currentDay+"'})";
6     int res=stmt.executeUpdate(query);
7     stmt.close();
```

Toch levert het uitvoeren van deze code een SQL-exceptie op. De foutmelding verbonden aan deze exceptie is '*General Error*' met de code 4010, hetgeen een *default*-waarde is. De *insert*-opdracht is correct opgesteld, de types van de in te voegen attributen zijn compatibel met de types van de databankattributen. Er is geprobeerd om alle weglatingstekens (symbool: ' ) te vervangen door \' maar ook dat leverde niets op. Een zoektocht naar eventuele oplossingen op *Internet* en in nieuwsgroepen leverde niets op.

De enige oorzaak die ik kan vermoeden is dat de JDBC-verbinding naar Access niet in staat is om een tuppel in te voegen waarvan de primaire sleutel een datumattribuut bevat. Een mogelijke oplossing voor dit probleem is om de primaire sleutel (*AandeelSleutel*, *Begin*) te verwijderen en te vervangen door een nieuw sleutelattribuut *NaamSleutel*; dit wordt in [GEMI98] meermaals toegepast.

De tabel *AandeelKBSNaam* wordt dan als volgt gedefinieerd:

```
Table AandeelKBSNaam:
      NaamSleutel  LONG
      AandeelSleutel LONG
      Naam  String
      Begin Date
      Einde Date
      Primary Key (NaamSleutel)
```

Het codefragment dient dan als volgt aangepast te worden:

```
1     stmt=con.createStatement();
1a    long nameKey=GenerateNewKey("AandeelKBSNaam");
2     query="INSERT INTO AandeelKBSNaam("+
2a          "NaamSleutel,AandeelSleutel,Naam,Begin)" +
3           " VALUES  ("+nameKey+
3a          "'"+stockKey+","+
4           "'"+newName+"',"
5           "{d '"+currentDay+"'})";
6     int res=stmt.executeUpdate(query);
7     stmt.close();
```

Deze aanpassing is uitdrukkelijk niet doorgevoerd en dit om volgende redenen:

- Het codefragment wordt enkel gebruikt in een klassemethode om de naam van een aandeel te wijzigen. Tot nog toe is bij het ingeven van koersgegevens geen enkele naamsverandering opgetreden.
- Nieuwsgroepgebruikers van *news:comp.databases.oracle* hebben me gegarandeerd dat het oorspronkelijke codefragment geen problemen zal opleveren in Oracle8.
- Ik ben het niet altijd eens met het invoeren van een loze sleutel ter vervanging van een primaire sleutel in een tabel. In [GEMI98] worden meermaals loze sleutels gebruikt om opzoekwerk in tabellen te vergemakkelijken maar Oracle8 heeft krachtige algoritmes om tabellen te indexeren volgens hun primaire sleutel.

## *6.3 Het memo-datatype in Access*

In het koersboek worden effecten binnen eenzelfde categorie alfabetisch gesorteerd op naam. Het attribuut *naam* van de entiteit *aandeel* is logischerwijze van het type *string*. Microsoft Acces ondersteunt strings tot een lengte van 255 karakters.
De eenvoudige bevraging '*SELECT naam FROM aandeel ORDER BY naam*' levert een geordende lijst aandeelnamen.
In de koerslijsten komen echter namen voor die langer zijn dan 255 karakters; in Access wordt dit opgevangen worden door het attribuut *naam* het type *memo* te geven. Hierdoor worden namen mogelijk met een lengte tot 65000 karakters. Probleem hierbij is dat Access kan niet sorteren op *memo*-velden.

Twee mogelijke oplossingen liggen voor de hand:

- Analoog met het attribuut *VolgendeCategorie* in de tabel *Categorie*, kan de volgorde van de aandelen binnen een categorie expliciet worden opgeslagen, maar dat zou bijzonder plaatsinefficiënt zijn. Daarenboven zouden nieuwe aandelen manueel[30] in de Access tabellen moeten worden ingevoegd, tezamen met de aanpassingen van de structuurtabellen. Elke fout zou leiden tot een inconsistente databank.
- De namen kunnen ongeordend worden opgevraagd door de elementaire query 'SELECT naam FROM aandeel' en dan manueel geordend worden. Praktisch is dit niet haalbaar, vermits het opvragen van de namen van aandelen in een koersboekcategorie gebeurt binnen een complexe reeks bevragingen die op zich al bijzonder veel tijd in beslag nemen. Het sorteren van namen neemt meer tijd in beslag dan verwacht vermits de elementen dan in een Java-*Vector* gestockeerd worden. Bij elke vergelijking in het sorteringsalgoritme treedt telkens een vertraging op ten gevolge van expliciete typeveranderingen.

Blijkbaar is er geen elegante oplossing zonder de testmodule grondig af te stemmen op een Access-databank, terwijl de beoogde databank toch Oracle8 is. In Oracle8 is de lengte van het type *varchar2* beperkt tot 4000 karakters, hetgeen zeker zal volstaan om elke aandeelnaam in te stockeren.
In overleg met dr. Buelens is gekozen om zolang met Access gewerkt wordt, de namen waar nodig verkort in de module te vermelden; dit zal slechts sporadisch voorkomen.

[30] Automatisch invoeren van nieuwe aandelen is niet mogelijk ten gevolge van het *update*-probleem, besproken in 6.2

# Hoofdstuk 7 : Informaticaplanning

Naar mijn mening is het informaticagedeelte van het SCOB-project vlot van start gegaan. De datastructuur die in hoofdstuk 5 is uitgewerkt en de koersmodule, besproken in hoofdstuk 6, staan centraal in de huidige werking van het project. Deze twee elementen vormen een stevige basis voor de verdere uitbouw van het project. In dit hoofdstuk wil ik een paar richtingen aangeven waarlangs het SCOB-project kan worden uitgebreid.

## *7.1 Programmeerprojecten*

Op dit moment is enkel de door mezelf ontwikkelde aandeelmodule beschikbaar voor de invoer van gegevens. Het komende jaar zullen nog een paar modules ontwikkeld worden die de invoer van andere gegevens mogelijk maken:

- **Obligatiemodule** – De eerste echte analyses die zullen worden uitgevoerd op de SCOB-databank zijn de vergelijkingen tussen de risicovolle en de risicoloze beleggingen, m.a.w. een vergelijking tussen de koersen van aandelen en die van obligaties. Tot nu is enkel een module ontwikkeld om aandeelkoersen in te geven, er moet dus een analoge module ontwikkeld worden om obligatiegegevens in te geven. Het relationele model hiervoor is reeds uitgewerkt in [GEMI98]. Voor de visualisatie van de koersboekstructuur kan de datastructuur uit de aandeelmodule rechtstreeks worden overgenomen. Naast de eigenlijke implementatie zal de programmeur zich voornamelijk moeten bezighouden met het ontwerp van een gebruiksvriendelijke *interface* want een obligatie is een complexer gegeven dan een aandeel; er moeten dus meer gegevens worden ingevoerd door de gebruiker.
- **Documentmodule** – Historicus Hans Willems heeft de afgelopen maanden de knipseldossiers uit het beursarchief doorgenomen. Hij heeft een exhaustieve inventaris van deze dossiers opgesteld in een tekstbestand [WILL99][31]. In [GEMI98] is een relationeel schema opgesteld waardoor deze inventaris kan opgenomen worden in de databank. Bedoeling van de documentmodule is om documentgegevens te kunnen invoeren in de databank, deze gegevens zijn dus ruimer dan de lijst uit [WILL99]. Elk document moet kunnen gekoppeld worden aan een bedrijf uit de databank. Om te vermijden dat elke document-vermelding uit [WILL99] domweg moet overgetypt worden, moet de module een geïntegreerde knip- en plakmogelijkheid hebben. Naast de invoer van documentgegevens moet ook een zoekfunctie geïmplementeerd worden die het mogelijk maakt om documentgegevens via Internet vrij te geven.
- **Bedrijfsmodule** – In juli 1998 heeft Jasper Nuyens zich als jobstudent bezig gehouden met een module die het mogelijk maakt om de evolutie van een bedrijf te visualiseren op basis van gegevens uit de SCOB-databank. Het resultaat van zijn werk is nogal middelmatig en zeker voor verbetering vatbaar. Daarenboven zijn nog niet alle benodigde gegevens ingevoerd in de SCOB-databank. Er is dus nood aan een invoermodule voor bedrijfsgegevens en in het bijzonder de evolutiegegevens van bedrijven zoals fusies, naamsveranderingen, opslorpingen, faillissementen …
  Het belang van deze module mag niet onderschat worden, want zoals eerder al gesteld zijn de bedrijfsgegevens essentieel voor de link tussen de document-historische data enerzijds en de financieel-economische data anderzijds.
- **FET-module** – Medewerkers van de *Financieel Economische Tijd* hebben voorgesteld om gratis de digitale koersgegevens uit hun databank ter beschikking te stellen. Op die manier kan het SCOB beschikken over alle koersgegevens van de Beurs van Brussel sinds 1984. Probleem is dat het SCOB geen rechtstreekse toegang krijgt tot de FET-databank, de gegevens zullen in ASCII-formaat per diskette geleverd worden. Deze gegevens moeten dus omgezet worden naar de SCOB-databank. Aangezien de gegevens in de FET-bestanden consistent geformatteerd en geïndenteerd zijn, kan deze omzetting automatisch gebeuren. De implementatie kan in PL/SQL, C of Java gebeuren.
- **Tijdreeks-vraagtaal** – Zoals gezegd in paragraaf 4.9 (p. 50) zal het SCOB nuttig gebruik kunnen maken van een SQL-variant met specifieke constructies voor tijdreeksbevragingen. Dergelijke vraagtaal is momenteel niet beschikbaar in Oracle8. Door een *parser* te laten communiceren met de SCOB-databank kan een bevraging in bv. de ITDM-vraagtaal worden omgezet naar een PL/SQL-bevraging in Oracle8.

[31] Appendix D bevat een uitreksel uit [WILL99].

## *7.2 Onderzoeksprojecten*

Naast het implementeren van verschillende invoer- en zoekmodules voor de SCOB-databank zijn er ook de meer theoretische onderzoeksonderwerpen. Deze projecten moeten met de verantwoordelijken van de verschillende werkgroepen van het SCOB worden uitgewerkt; afhankelijk van de analyses die zij wensen uit te voeren en van de aanwezige data in de SCOB-databank kunnen deze projecten verder ingevuld worden.

- **Datamining** – De SCOB-databank is voornamelijk bedoeld voor onderzoek van historische gegevens. Uiteraard zijn de financieel-economen op de hoogte van de gebruikelijke analyses die ze willen uitvoeren. De meeste van die analyses zullen uitvoerbaar zijn via SQL-bevragingen maar eens de SCOB-databank een representabele hoeveelheid gegevens omvat, kan ze ook dienst doen als bron voor *datamining*.
  Een voor de hand liggende vorm van *dataming* is het zoeken van gelijkenissen tussen tijdreeksen: «*Zijn er onverwachte gelijkenissen tussen de koersen van totaal verschillende aandelen?*»
  Meer gevorderde vormen van tijdreeks-*mining* zijn ook mogelijk. Bv. het antwoord op de vraag «*Zit er een patroon in de koersdalingen en -stijgingen gedurende een bepaalde periode?*»
  Dit kan als volgt gebeuren:
  - Pas *datasmoothing* toe op de koers-tijdreeksen.
  - Bereken de afgeleide functie.
  - Tussen al deze afgeleide functies (tijdreeksen) worden dan gelijkenissen gezocht.

  De berekende afgeleides kunnen ook gebruikt worden voor het vinden van associaties tussen bijvoorbeeld de financiële toestand van een bedrijf en de evolutie van de aandeelkoersen.
  De *Gouden Graal* onder de koersanalyses is de zoektocht naar een classificatie die in staat is om te voorspellen of een beurseffect winstgevend is of niet. Prof.dr.ir. Arie Weeren van Ufsia is met een soortgelijk onderzoek bezig, waarbij neurale netwerken getraind worden om wisselkoersen tussen buitenlandse munteenheden te voorspellen.
  Het onderzoek naar de mogelijkheden van de genoemde *datamining*-projecten omvat zowel het onderzoek naar bruikbare en onderhoudbare datastructuren als het opstellen en testen van *datamining*-algoritmes.

- **Zoekproblemen** – In [GEMI98] wordt het relationele schema voor een personenmodule gepresenteerd aan de hand van een tiental tabellen, waarvan de meeste tabellen normale gegevens stockeren. Er is ook een tabel *Relaties*, die de familiale banden tussen personen als volgt weergeeft:

| **Relaties** | |
|---|---|
| Persoon | Verwijzing naar een uniek persoon. |
| Relatie | Verwijzing naar een uniek persoon. |
| Band | Type relatie tussen de twee personen.<br>Bv.: *Getrouwd met*, *Broer van*, *Vader van* |

**Tabel 14:** structuur van de tabel *relaties*.

  Deze tabel zal een groot aantal relaties bevatten, dus een SQL-bevraging op zich moet al zo efficiënt mogelijk worden opgesteld, maar een bijkomend probleem is dat de tabel nooit volledig kan ingevuld worden:

  > Stel dat persoon *A* een broer is van persoon *B* en dat deze relatie is ingevoerd in de databank, meteen kan dan worden ingevoerd dan *B* een broer (of zus) is van *A* maar stel daarenboven dat *B* twee kinderen heeft (*C* en *D*); dit wordt ingegeven door (B,C, *Vader_van*) en (B,D, *Vader_van*) te stockeren in de tabel *Relaties* eventueel tesamen met de omgekeerde *Zoon_van* relatie maar het is onhaalbaar om bijvoorbeeld ook (A,D, *Nonkel_van*) te stockeren. Toch moet deze relatie opvraagbaar zijn uit de databank.

  Om deze vraagstelling mogelijk te maken zonder elke mogelijke relatie in de databank te stockeren, zal een vorm van logische inferentie aangeleerd moeten worden aan de programmatuur van de databank. Een directe oplossing is het koppelen van de databank aan *PROLOG* maar er moet zeker ook gezocht worden naar een oplossing in het vakgebied van de artificiële intelligentie waarbij de klemtoon ligt op efficiënte zoekmethodes en goede heuristieken voor deze zoektochten.
  Verder moet ook bekeken worden hoe de instellingen van Oracle8 geconfigureerd kunnen worden voor een optimale informatieverwerking.

## *7.3 Mogelijke uitbestedingen*

Op het studieprogramma van eerste licentie informatica op UIA komt het vak *Practicum Programmeren* voor. De student dient hiervoor een concreet project uit te werken door middel van specificatie, ontwerp, implementatie en documentatie. De resulterende programmatuur moet concrete toepassingen kennen.
Samen met dr. Buelens is het volledige SCOB-project geanalyseerd. Na een controle van de haalbaarheid van de verschillende modules uit [GEMI98] zijn drie modules uit paragraaf 7.1 geschikt bevonden en ingediend als onderwerp voor *Practicum Programmeren* voor het academiejaar 1999-2000; het betreft hier de obligatie-, de documenten- en de bedrijfsmodule. Hierbij moet men goed voor ogen houden dat indien een student kiest voor één van deze onderwerpen de uiteindelijke programmatuur pas tegen juni 2000 zal afgewerkt zal zijn.

De projecten uit paragraaf 7.2 zijn meer onderzoeksgericht en vergen minder programmeerwerk. Daardoor zijn ze niet geschikt als jaarproject in de eerste licentie maar wanneer ze worden uitgebreid met de opdracht om lectuur over de onderwerpen door te nemen en samen te vatten, zijn ze volgens mij geschikt als thesisonderwerp voor informaticastudenten uit de tweede licentie. Voor dergelijk onderzoek is echter een uitgebreide databank nodig en tevens een begeleider die tegelijk op de hoogte is van de *datamining*-onderwerpen en van de SCOB-databank. Momenteel beschikt het SCOB over geen van beide, dus kunnen deze onderwerpen ten vroegste in het academiejaar 2000-2001 voorgesteld worden als thesisonderwerp.

De afdeling informatica van de *Karel de Grote* hogeschool is een erkend *Oracle Training Centre*. De heer De Bruyn, verantwoordelijke van deze afdeling, heeft zich bereid verklaard om laatstejaarsstudenten de mogelijkheid te bieden om mee te werken aan het SCOB-project en hen zo praktijkervaring te laten opdoen. Dergelijke medewerking van hogeschoolstudenten is van vrij korte duur (10 weken tijdens de maanden april, mei en juni) en het moet zeker nog bekeken worden of dit kan bijdragen tot de verdere uitwerking van het SCOB-project.

Naast het uitbesteden van programmeer- en onderzoekswerk aan informaticastudenten zal de toekomstige informaticamedewerker zelf de komende maanden nog heel wat uitbreidingen moeten doorvoeren aan de SCOB-databank; het betreft voornamelijk het verder aanpassen van de databank voor invoer van coupon- en dividendgegevens en de omzetting van de Access-databank naar Oracle8 op een SUN-platform.
Daarenboven is er geen zekerheid dat studenten ook effectief zullen ingaan op de onderwerpen voor jaarprojecten en thesissen, waardoor de SCOB-informaticus zelf de nodige projecten zal moeten uitwerken.

# **Hoofdstuk 8 :** Keuze van het DBMS

Het *SCOB* wilde een databank opstarten waarin voorlopig slechts een deel[32] van de gegevens van aandelen, bedrijven en personen kan worden opgeslagen. De beslissing tot aankoop van een DBMS was dan ook één van de eerste opdrachten van de deelgroep informatica; daarom heb ik in samenwerking met dr. Gemis een vergelijkende studie gemaakt van verschillende beschikbare DBMS'en. De resultaten van deze analyse zijn gebaseerd op artikels (o.a. [BYTE9709], [BYTE9806] en [STON97[), commerciële informatie, mails van werknemers van de verschillende databankproducenten en discussies in *newsgroups*. Dit hoofdstuk geeft hiervan een samenvatting. Ik verplicht me er meteen toe te vermelden dat niet alle gegevens even gefundeerd en accuraat zijn, omdat het bijzonder moeilijk is om vooroordelen t.o.v. bepaalde standpunten te doorprikken en correcte informatie te verkrijgen.
De integrale tekst van dit hoofdstuk is ter correctie overgemaakt aan elk van de producenten. Blijkbaar achtten enkel Informix en Oracle het nodig om effectief correcties voor te stellen; deze verbeteringen zijn dan ook ingevoegd in de bespreking.

## *8.1 Inleiding*

De huidige trend in de databankwereld heet *'universal database'*. Met universeel wordt bedoeld dat elke databankapplicatie elk datatype aankan op gelijk welke schaal. Voor gebruikers mag er geen verschil merkbaar zijn tussen de klassieke attribuuttypes en de meer complexe datatypes zoals beelden, tijdreeksen, geluidsfragmenten en documenten.
Niet alle aspecten van universele databanken zijn van belang voor deze analyse, zo zal de SCOB-databank op korte en middellange termijn zeker geen multimediale elementen moeten bevatten; uiteraard is hiermee dan ook geen rekening gehouden.

Voor het SCOB zijn voornamelijk volgende aspecten van belang:
- efficiënte en gebruiksvriendelijke tijdreeksen
- mogelijkheid tot datatoegang via Java
- prijs van de databank, van de support en van het onderhoudscontract
- beschikbaarheid op een Sun-platform

De onderzochte DBMS'en zijn in alfabetische volgorde: *IBM DB2 UDB*, *Informix Universal Server*, *Microsoft SQL/Server 7.0*, *Oracle8* en *Sybase Adaptive Server*.

## *8.2 DB2*

IBM haalt in zijn promotie voor *DB2 Universal Server 5.0* stevig uit naar de concurrentie maar de argumentatie is niet echt overtuigend, veelal doet die zelfs niets terzake.

Een greep uit de irrelevante argumenten t.o.v. de concurrentie:
- Oracle8: Oracle kan slechts 10000 gebruikers aan, DB2 haalt een maximum van 64000.
- Informix: IBM vergelijkt enkel met Informix Illustra. «*[…] het Illustra product is stopgezet. Illustra was in '93 de eerste commerciële ORDBMS op de markt. Nadat Illustra eind '95 door Informix werd opgekocht, werd het gemerged met de bestaande Informix database engine. Op 6/2/96 lanceerde Informix zijn Universal Server. Dit product werd in oktober '97 in de product simplification gerenamed naar IDS/universal data option.*»[33]
- SQL/Server: IBM verkoopt meer NT-software dan Microsoft.

Een belangrijke reden waarom SCOB niet gekozen heeft voor IBM, is dat IBM nooit gereageerd heeft op offerte-aanvragen of op aanvragen om meer informatie. Dit is echter niet de enige reden: IBM is geen databank-specialist; hun product mag dan wel gezien worden, het bevat toch de nodige tekortkomingen.
IBM mag dan nooit een mogelijke keuze geweest zijn van het SCOB toch is DB2 opgenomen in deze vergelijkende analyse.

[32] Voorlopig is het doel om per aandeel maandelijks één koers te stockeren in de databank.
[33] Luc Van de Velde – Informix - lucvdv@informix.com

Pluspunten:
- DB2 is een consistent DBMS, betrouwbaar en gemakkelijk te onderhouden.
- IBM heeft nog steeds een innovatief karakter, zo kwam IBM als eerste met Java- en JDBC-ondersteuning.
- DB2 heeft een zeer krachtige query optimizer die kan draaien op een parallelle architectuur met 320 processoren.

Minpunten:
- De user-interface software is niet altijd even consistent. De user-interfaces overstijgen ruim de verouderde command-line interfaces van eerdere versies maar grafisch zijn ze toch nog niet helemaal in orde.
- IBM legt zich steeds meer toe op het internet-gebeuren waardoor uitbreidingen in de richting van complexe datatypes een lagere prioriteit krijgen; verbeterde ondersteuning voor tijdreeksen valt volgens buitenstaander ook onder deze lagere prioriteit.
- DB2 is niet platformonafhankelijk: DB2 op MVS, DB2 op VSE (vroeger SQL/DS), DB2 op AS400 (vroeger DB400), DB2 op NT, DB2 op OS2, DB2 op AIX, DB2 parallel edition, DB2 Universal Database zijn totaal verschillende producten.
- Naamgeving is nog steeds een belangrijk tekort. DB2 laat slechts 18 karakters toe voor naamgeving van tabellen, gebruikers en programmacode en dat is niet altijd voldoende.

### *8.3 Informix*

Volgens kenners staat Informix tezamen met Oracle in de top-2 van ‘s werelds beste DBMS’en, maar welke van de twee op de eerste plaats staat, is niet eenvoudig te bepalen. Veel hangt af van de eisen die men stelt bij de keuze tussen de twee systemen. Als je de feiten bekijkt lijken de meeste databanken een afspiegeling te zijn van Informix-technologie. Eind ’95 nam Informix het bedrijf Illustra over omdat Informix inzag dat ORDBMS de toekomst was. Half ’96 was de Illustra-technologie geïntegreerd in *Informix Universal Server*. Ongeveer drie weken later werd *Oracle 7.3 Server* –zonder wijzigingen– van naam veranderd naar *Universal Server*.

De meeste databankgebruikers zijn het erover eens dat Informix de beste technologieën in handen heeft, meer nog: het databankmodel is veel doorzichtiger maar de beste technologie kan blijkbaar niet opboksen tegen een gevestigde waarde zoals Oracle.

Pluspunten:
- De schaalbaarheid van Informix scoort het beste op de databankmarkt; zowel op een monoprocessor-architectuur als op multiprocessor-architecturen of netwerkclusters. De databank is schaalbaar van megabytes tot vele terabytes.
- Informix staat veel verder met zijn ORDBMS dan gelijk welke concurrent. De door de gebruiker gedefinieerde datatypes worden echt geïntegreerd op serverniveau.
- De Informix datablades zijn echte cartridges die extern kunnen worden toegevoegd.
- De installatie van Informix vereist veel minder externe instellingen dan bij Oracle. Het onderhoud van een draaiende Informix versie verloopt gemakkelijker dan Oracle.
- Qua Datawarehousing en Internet-technologie lijkt (is?) Informix marktleider.

Minpunten:
- Methodes voor procedureel onderhoud van de databank kunnen niet in Java geïmplementeerd worden, bij Oracle kan dit in C, Java, PL/SQL. Informix heeft ondertussen wel een beta-Java-ondersteuning.
- Informix heeft geen duidelijke lange-termijn-visie. Het blijft onduidelijk welke richting Informix uit wil met zijn technologie.

### *8.4 Microsoft SQL/Server*

Microsoft SQL/Server is buiten beschouwing gelaten omdat het SCOB op voorhand al gekozen had voor een SUN-platform, terwijl SQL/Server enkel de platformen Intel, Alpha met Windows (NT & 9x) ondersteunt. Op het moment dat deze tekst werd geschreven, was SQL/Server 6.5 verkrijgbaar en was versie 7.0 in het beta-stadium. SQL/Server 6.5 vergelijken met Oracle of Informix is oneerlijk omdat SQL/Server eerder behoort tot de klasse van kleine en middelgrote DBMS’en, voor single-user en KMO-omgevingen; vergeleken met de grote ORDBMS’en blijft SQL/Server het kleine broertje.
Microsoft is geen concurrentie voor de andere databankproducenten. Toch verblinden ze potentiële databank-gebruikers met blitse interfaces.

### *8.5 Oracle*

Oracle is dé gevestigde waarde in de databankwereld. Hun technologie is meestal net niet up-to-date, maar tegelijk ook zelden verouderd.

Pluspunten:

- Oracle kan zich beroepen op een stevig marktaandeel waardoor continuïteit van support zo goed als zeker gegarandeerd kan worden.
- Een standaard Oracle installatie is één geheel, dus elke applicatie kan ten volle gebruik maken van de capaciteiten van de databank.
- De *development tools* van Oracle8 krijgen een goede score. **Pluspunten**: *Web-development*, *top-down* & *bottom-up-development*, *object-builders*, *Java-components*.
- Technische ondersteuning: geregistreerde gebruikers kunnen steeds terugvallen op een gespecialiseerd team terwijl dit bij Informix soms te wensen overlaat.

Minpunt:

- De installatie en het onderhoud van Oracle8 is verre van vanzelfsprekend.

### *8.6 Sybase*

Het Sybase DBMS promoot het gebruik van verschillende aparte datastores. Daarmee verschilt Sybase van alle concurrenten. Om die reden wil ik hier toch even dieper op ingaan. Er wordt vanuitgegaan dat als de data gespreid wordt in verschillende stores aan serverzijde, zo'n store dan veel gespecialiseerder en efficiënter kan omgaan met de verschillende soorten data. Het gebruik van verscheidene datastores betekent het verplicht invoeren van verschillende interfaces. Hetgeen op zijn beurt dan weer een lagere performatie betekent omdat er geen directe communicatie en datatransferts mogelijk zijn tussen de datastores. Elke datatransfert moet vertaald worden door de interface van de zogenaamde *Component Integration Layer* die alle interfaces omvat tussen de verschillende datastores. Voor eenvoudige bevragingen –die enkel gegevens gebruiken uit één datastore– zal de efficiënte benutting van dataspecialisatie in datastores het interfaceprobleem wegwerken omdat met buffers heel wat interface-vertalingen zo goed als optimaal verlopen. Wanneer echter een complexe bevraging data puurt uit meerdere datastores, verhinderen de interfaces een optimale werking.

Het interfaceprobleem is te omzeilen wanneer de databankadminstrator een goed inzicht heeft in de opbouw van alle interface-lagen, zodat door middel van directe communicatie met de datastores heel wat interface-bottlenecks omzeild kunnen worden. Inzicht in de interface-lagen is echter niet voor de hand liggend en de meer directe communicatie met de databank valt volledig in de categorie van 'vuil programmeren', voor elk efficiëntieprobleem moet een specifiek algoritme ontwikkeld worden.

Sybase heeft zelf geen technologie om op eenvoudige wijze met tijdreeksen te werken. Enkel het product *Fame* biedt deze mogelijkheid, maar de hoge kostprijs (4 miljoen BEF voor *FAME* en 1.5 miljoen BEF voor de connectie met de rest van de databank) plaatst hen buiten alle concurrentie.

Uiteraard heeft Sybase ook positieve eigenschappen: hun *mobile computing* technologie is bijzonder ontwikkeld maar dit is van weinig belang voor het SCOB.

### *8.7 Conclusie*

In de voorgaande paragrafen zijn de belangrijkste plus- en minpunten van de verschillende DBMS'en zo objectief mogelijk uiteengezet. Toch mag de kwaliteit van een product niet beoordeeld worden op het aantal plus- en minpunten; daarenboven bestaat er niet zo iets als objectieve kwaliteitsbeoordeling.
Wat het SCOB betreft zijn er de twee grote (klassieke !) uitschieters, nl. Informix en Oracle. Beide bieden voldoende functionaliteiten om in aanmerking te komen voor aankoop.

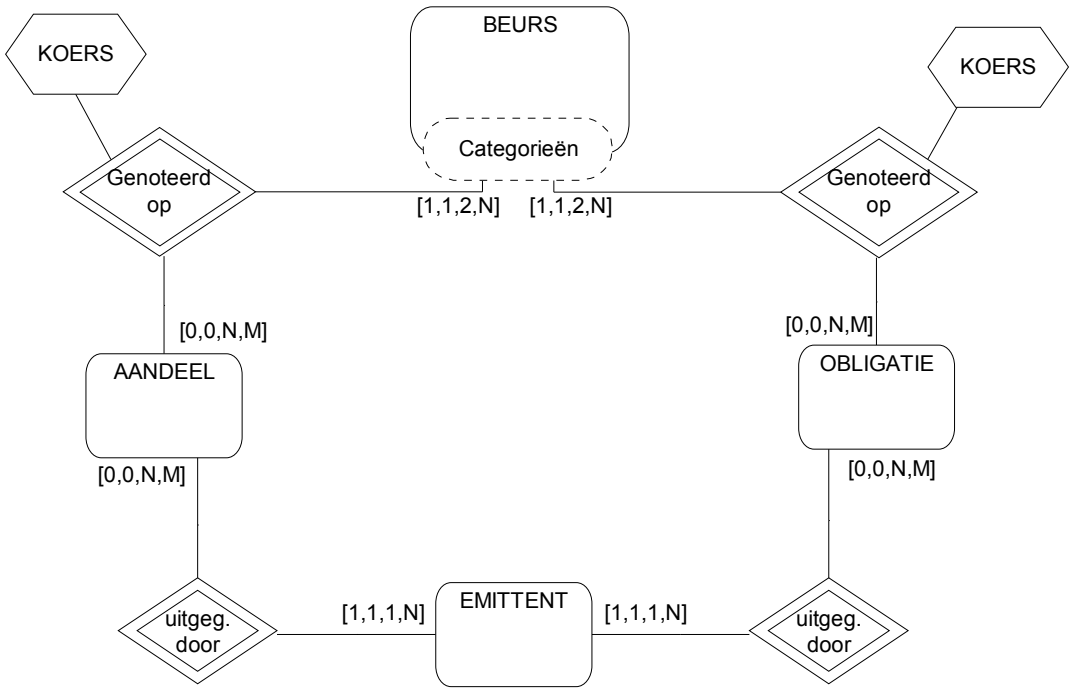
- Men heeft beslist om de Oracle databank aan te kopen om volgende redenen:
- Pas na aanzienlijke financiële toegevingen van Informix lagen de prijzen van Oracle en Informix in dezelfde categorie.
- UIA, RUCA en Ufsia hebben al eerder Oracle databanken aangekocht en geïnstalleerd. De verschillende rekencentra hebben dus meer ervaring en voeling met Oracle.
- Hoewel Informix economisch weer solvabel is, is nog niet helemaal zeker wat de toekomstperspectieven van het bedrijf zijn.

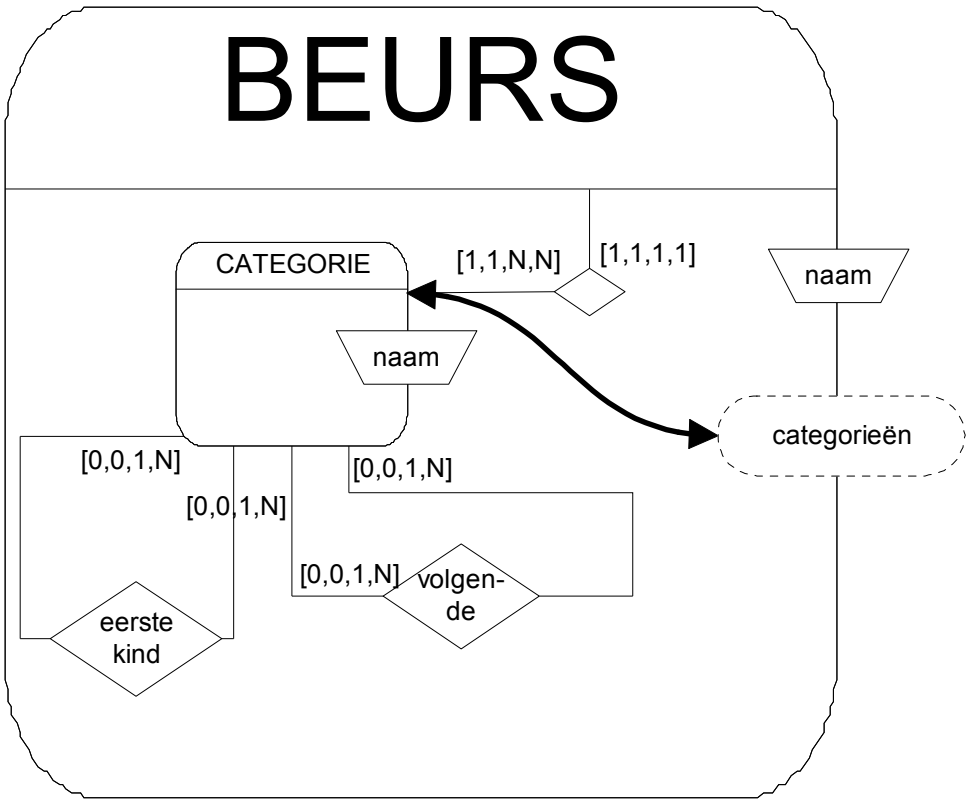
# Appendix A. Het LACER-diagramma van de SCOB-databank

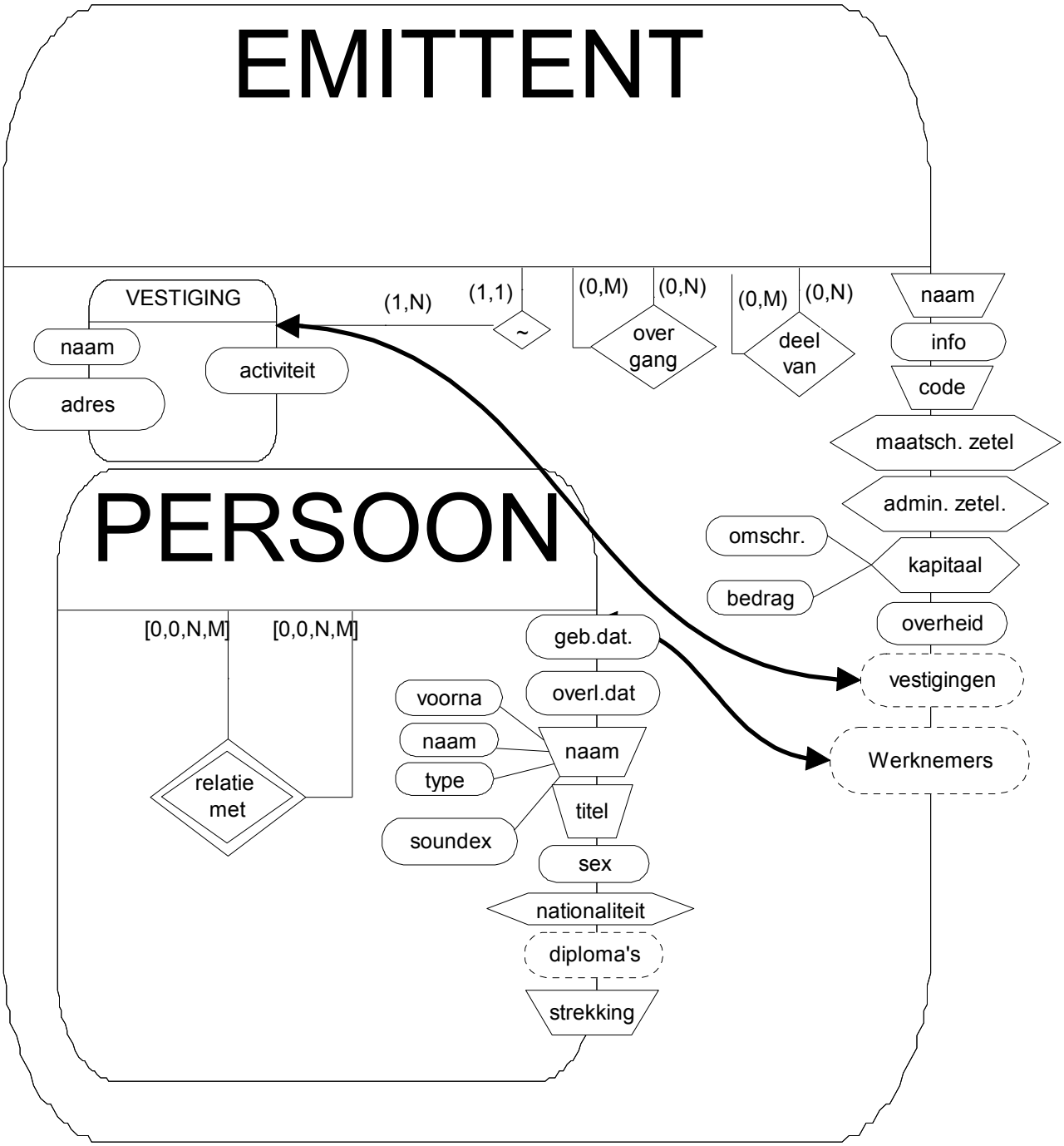
De LACER-diagramma's uit dit appendix dienen ter vervanging van de UML-diagramma's uit het werkdocument [GEMI98]. Het diagramma modelleert slechts een klein aantal entiteiten en relaties en dit om volgende redenen:

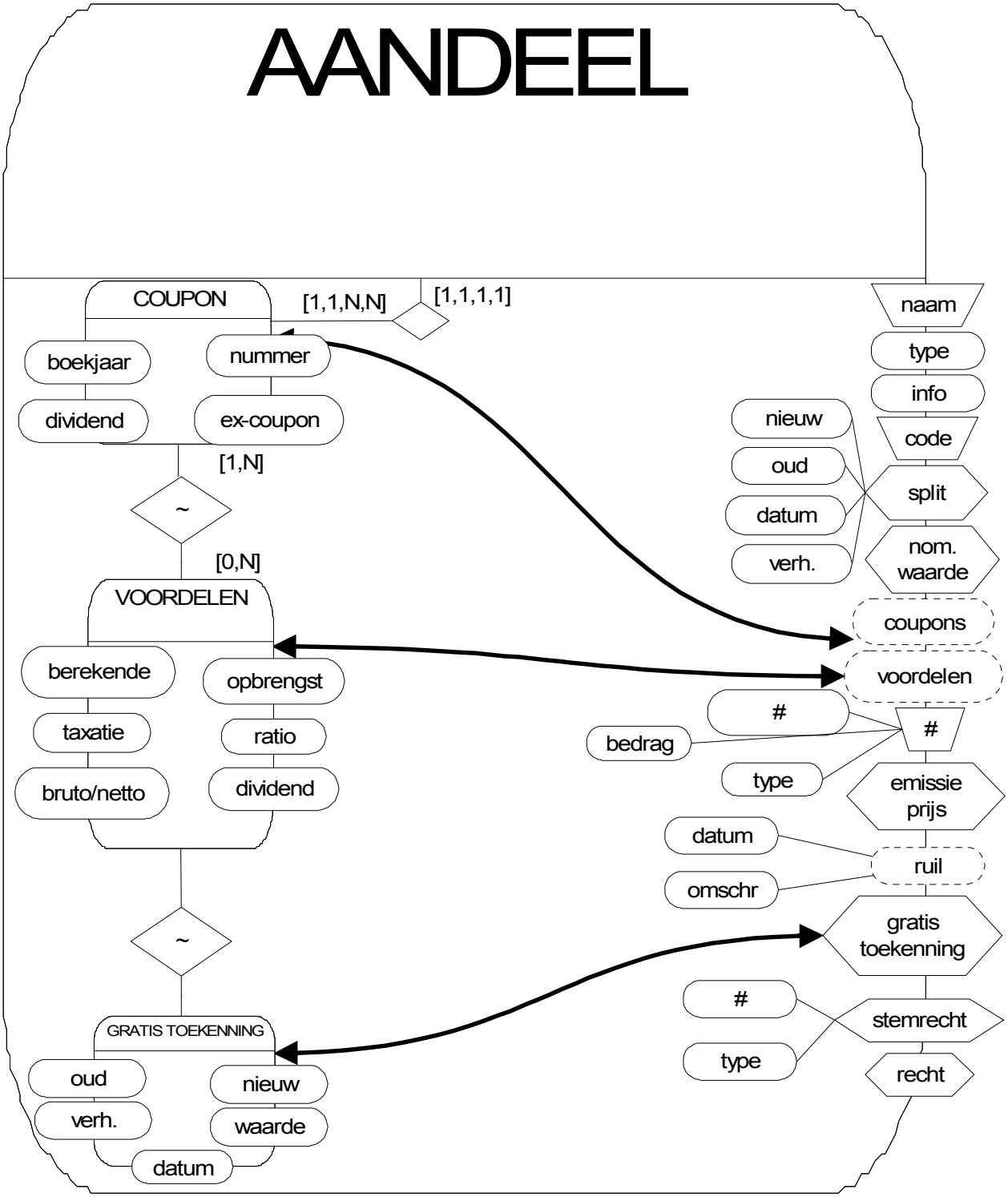
- Samen met dr. Buelens heb ik [GEMI98] doorgenomen en hebben we bepaald voor welke tabellen het haalbaar is om ze effectief in te vullen. Alle andere tabellen zijn wel behouden in het werkdocument maar niet opgenomen in het LACER-diagramma.
- Sommige tabellen in [GEMI98] zijn te gedetailleerd. Onder invloed van de historici heeft dr. Gemis de structuur van de documentmodule zeer grondig uitgewerkt. Deze structuur is correct maar niet haalbaar binnen het project. Welke structuur dan wel gehanteerd zal worden, hangt af van de informatie die de heer Willems kan puren uit de archiefdozen en van de gegevens die de historici willen analyseren. Het contact met de historici verloopt echter stroef; bijgevolg is de documentstructuur nog niet herzien. Daarenboven kan een document aan zowat elke entiteit gekoppeld worden want er zijn documenten over bedrijven, overheden, beurzen, aandelen, obligaties, overnames, ... Eens Oracle gebruikersklaar is, zal gecontroleerd moeten worden of PL/SQL hiervoor een flexibele oplossing biedt. Het relationele model bepaalt hier ten dele het LACER-diagramma.
- Bepaalde aspecten uit de miniwereld zijn nog niet gemodelleerd. Zoals al meermaals vermeld in dit document zullen een viertal TEW-studenten zich het komende jaar gaan bezighouden met bepaalde analyses op het beursarchief. De resultaten hiervan zullen gestockeerd worden in het SCOB-databank maar de structuur hiervoor kan nog niet worden vastgelegd.

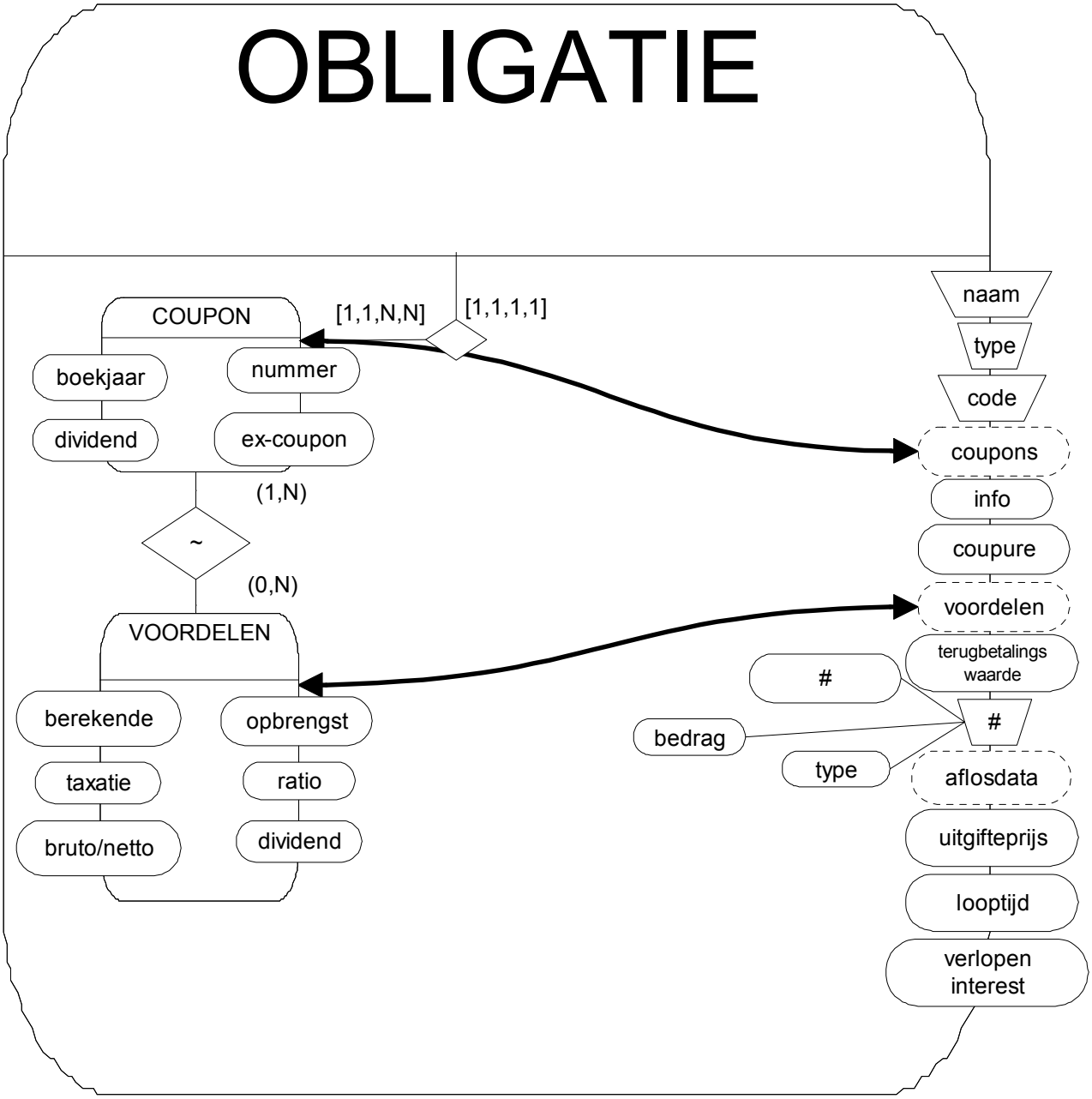
Het eerste diagramma is het meest abstracte model van de beurs miniwereld, alle daaropvolgende diagramma's zijn gedetailleerde versie van de entiteiten uit dit diagramma.




EMITTENT
VESTIGING
naam
adres
activiteit
(1,N)
(1,1)
~
(0,M)
(0,N)
over gang
(0,M)
(0,N)
deel van
naam
info
code
maatsch. zetel
admin. zetel.
kapitaal
overheid
vestigingen
Werknemers
omschr.
bedrag
PERSOON
[0,0,N,M]
[0,0,N,M]
relatie met
geb.dat.
overl.dat
naam
titel
sex
nationaliteit
diploma's
strekking
voorna
naam
type
soundex


AANDEEL
COUPON
[1,1,N,N]
[1,1,1,1]
boekjaar
nummer
dividend
ex-coupon
[1,N]
~
[0,N]
VOORDELEN
berekende
opbrengst
taxatie
ratio
bruto/netto
dividend
~
GRATIS TOEKENNING
oud
nieuw
verh.
waarde
datum
naam
type
info
code
nieuw
oud
split
datum
verh.
nom.
waarde
coupons
voordelen
#
#
bedrag
type
emissie
prijs
datum
ruil
omschr
gratis
toekenning
#
stemrecht
type
recht


OBLIGATIE
COUPON
[1,1,N,N]
[1,1,1,1]
boekjaar
nummer
dividend
ex-coupon
(1,N)
~
(0,N)
VOORDELEN
berekende
opbrengst
taxatie
ratio
bruto/netto
dividend
naam
type
code
coupons
info
coupure
voordelen
terugbetalings waarde
#
#
bedrag
type
aflosdata
uitgifteprijs
looptijd
verlopen interest

# Appendix B. PL/SQL code voor Oracle

De *CHECK*-clausule uit SQL-fragment 8 op p. 39 wordt niet ondersteund door PL/SQL van Oracle8. Oracle8 ondersteunt wel SQL-*triggers*, dus de *CHECK*-clausule moet omgezet worden naar een *trigger*.
Dit kan als volgt:

```
CREATE TRIGGER AandeelNaam_IntervalDuplicatie
AFTER INSERT OR UPDATE ON AandeelNaam
DECLARE
      valid:INTEGER
BEGIN
      SELECT 0
      INTO valid
      FROM DUAL
      WHERE NOT EXISTS (SELECT A.AandeelNaam
                        FROM AandeelNaam A, AandeelNaam B
                        WHERE A.AandeelSleutel=B.AandeelSleutel
                              AND A.AandeelNaam=B.AandeelNaam
                              AND A.Start<B.Einde
                              AND A.Einde<B.Start
                              AND A.rowid!=B.rowid
                        );
      EXCEPTION
            WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20001,'Interval-duplicaten gevonden');
END;
```

De definitie van deze *trigger* is bijzonder Oracle8-afhankelijk en vraagt dus enige verklaring:

- De binnenste *SELECT*-opdracht controleert of er intervalduplicaten voorkomen, door eerst een *join* tussen de tabel en zichzelf uit te voeren. Hierdoor wordt ook elk tuppel met zichzelf verbonden hetgeen altijd leidt tot intervalduplicatie. Oracle8 voegt echter aan elke tabel rij impliciet het attribuut *rowid* toe. Dit attribuut is voor elke rij uniek over de gehele databank. De controle `A.rowid!=B.rowid` vermijdt dus dat aan zichzelf gekoppelde rijen in rekening worden gebracht.
  Wanneer de binnenste *SELECT*-opdracht een niet-leeg resultaat teruggeeft dan is er sprake van interval-duplicatie.

- In de buitenste *SELECT*-opdracht komt de tabel *DUAL* voor. Deze tabel is pro-forma gedefinieerd in Oracle8 en dient als selectietabel in *SELECT*-opdrachten die eigenlijk geen *FROM*-gedeelte nodig hebben, zoals de volgende *SELECT*-opdracht: `SELECT power(4,3) FROM DUAL`

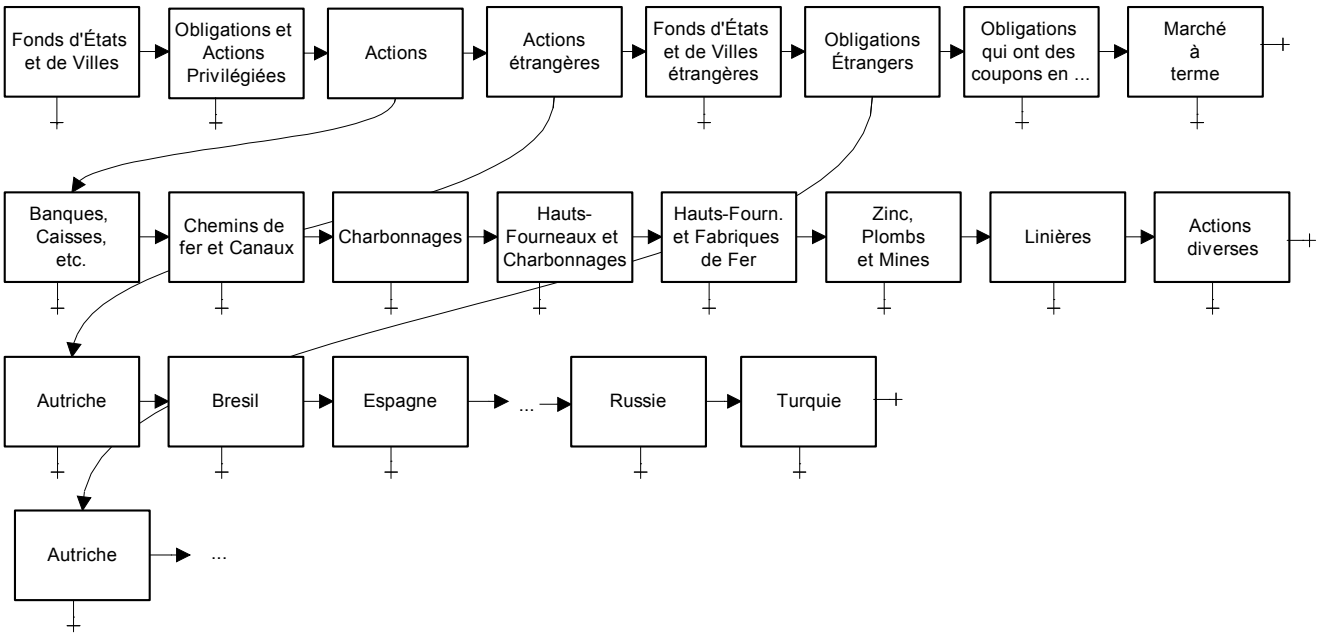
- Een *SELECT…INTO*-opdracht is de PL-SQL versie van de klassieke *SELECT*-opdracht waarbij het resultaat kan opgeslagen worden in een variabele. In dit geval wordt de waarde 0 opgeslagen in de variabele *valid*, tenminste wanneer het *WHERE*-gedeelte geldt. In het andere geval wordt de exceptie *NO_DATA_FOUND* gegenereerd, d.i. wanneer de binnenste *SELECT*-opdracht een intervalduplicaten heeft gevonden.

De andere *CHECK-clausules* uit hoofdstuk 4 kunnen analoog worden omgezet naar Oracle8-*triggers* in PL/SQL.

# Appendix C. Het koersboek

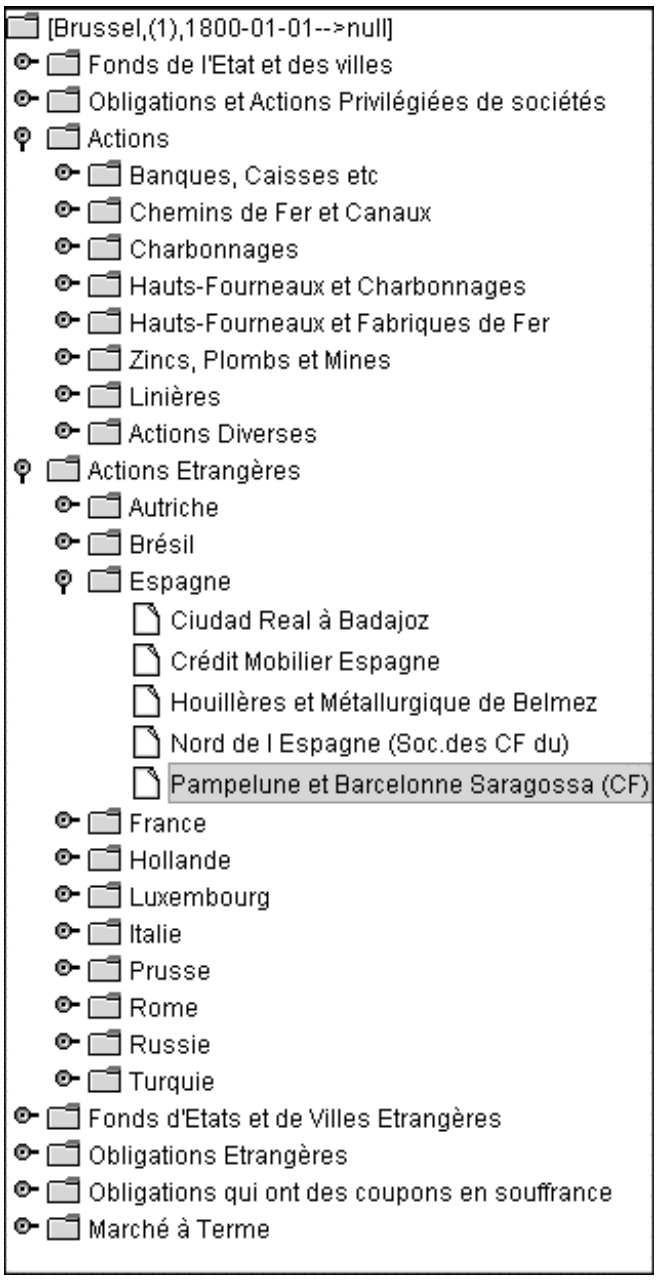
De modellering van de categorie-indeling in hoofdstuk 5 is het best te illustreren aan de hand van een concreet voorbeeld. Dit appendix bevat daarom het volgende:

- een kopie uit het koersboek van de Beurs van Brussel. De gekozen koersdag is 4 januari 1876. De kwaliteit van deze kopie is niet al te best maar dit is te wijten aan het feit dat de koersboeken van goede kwaliteit niet kopieerbaar zijn zonder ze te beschadigen. Daarom is een koersboek genomen waarvan de staat het toeliet om het op een kopieermachine te leggen, m.a.w. een koersboek dat onherstelbaar vernield was.
- de bijbehorende datastructuur (Figuur 33): de koersboekstructuur is hierbij gemodelleerd volgens de datastructuur uit paragraaf 5.2.
- een afdruk van de categorieboom (Figuur 34): De afdruk is een schermafdruk van de koersmodule.




**Figuur 33:** bijbehorende datastructuur.

De categorie «*Marché à Terme*» komt niet voor in de kopie van het koersboek; dit is te wijten aan het feit dat er slechts twee maal per maal koersen genoteerd staan in deze categorie.




**Figuur 34:** bijbehorende schermafdruk.

# Appendix D. Het koersboekInventaris van de knipseldossiers

Een knipseldossier is een map waarin alle beschikbare publicaties over een specifiek bedrijf worden bijgehouden. De knipseldossiers in het SCOB-archief zijn verdeeld over 87 dozen. Elke doos bevat een reeks mappen met documenten met informatie over bedrijven. Voor elk bedrijf uit het archief zijn dus één of meer mappen aangelegd. Het document [WILL99] is als volgt ingedeeld: per doos wordt vermeld over welke bedrijven informatie is opgenomen en per bedrijf wordt vermeld hoeveel mappen er zijn en over welke periode ze informatie bevatten. Onmiddellijk na da mapinformatie volgt een lijst van de documenten die zich in de mappen bevinden.
De rest van dit appendix is een uitreksel uit [WILL99].

**DOOS I**

*Banque de la société générale de Belgique*

- Map 1: 04/1944 – 12/1958
- Map 2: 01/1966 – 07/1968
- Map 3: 09/1968 – 12/1971
- Map 4: 01/1972 – 12/1974
- Map 5: 01/1975 – 12/1978
- Map 6: 01/1979 – 04/1982
- Map 7: 04/1982 – 12/1985
- Map 8: 01/1986 – 12/1987

BEVAT:

- SOCIETE GENERALE DE BANQUE, Rapport spécial du conseil d›administration aux actionnaires, rapport des commissaires-réviseurs, réviseurs d›entreprises (jaarverslag), 10/1965.
- SOCIETE GENERALE DE BANQUE, Informations de la société générale de banque, 9/1967.
- SOCIETE GENERALE DE BANQUE, Bulletin économique de la Société générale de banque, 11/1967.
- SOCIETE GENERALE DE BANQUE, Offre en souscription publique, 4/1969. (prospectus)
- SOCIETE GENERALE DE BANQUE, Offre en souscription publique, 9/1975. (prospectus)
- SOCIETE GENERALE DE BANQUE, Offre en souscirption publique, 5/1982. (2-talige prospectus, zeer uitgebreid)
- SOCIETE GENERALE DE BANQUE, Offre en souscription publique, 4/1985. (2-talige prospectus, zeer uitgebreid)
- GENERALE DE BANQUE, Offre en souscription publique, 4/1986. (2-talige prospectus)
- BBL (STUDIEDIENST FINANCIËLE ANALYSE), Générale bank, 4/1986
- GENERALE DE BANQUE, Augmentation de capital réservée exclusivement aux membres du personnel, 10/1987.
- Jaarverslagen: 1977 B 1984 + 1986 (zeer uitgebreid)

*Société Générale de Belgique*

- Map 1: 11/1942 – 11/1949
- Map 2: 1/1950 – 12/1959
- Map 3: 1-1960 – 12/1965

BEVAT:

- SOCIETE GENERALE DE BELGIQUE, Augmentation du capital social, 12/1945
- ASPECTS ECONOMIQUE ET FINANCIER, La bourse et l›investissement: Société générale de Belgique, 4/1952.
- SOC. GEN. DE BEL., Vente par souscription de 83.600 parts de réserve de 1.250 fr. nominal, 3/1955. (prospectus)
- KREDIETBANK, Documentation financière: Soc. Gén. De Bel., 6/1961.
- Les sociétés à portefeuille: Soc. Générale de Belgique, 7/1964.

# Appendix E. Schermafdruk van de invoermodule

# Gebruikte afkortingen

ADT Abstract Datatype: een datatype waarbij de data wordt afgeschermd van de gebruiker. Veranderingen aan een ADT kunnen enkel gebeuren via ingebouwde methodes. Hierdoor worden implementatie en gebruik van het datatype strikt gescheiden gehouden.

DBMS Database Management System: Het geheel van databankprogrammatuur.
De belangrijkste taken en mogelijkheden van een DBMS zijn:
- databank opbouwen en onderhouden
- bevraging en modificatie van data
- transactie-management en synchronisatie
- backup- en recovery

ERD Entity-Relationship Diagram: strikt genomen is een ERD enkel een diagramma van het ER-model maar de term wordt ook veelvuldig gebruikt voor diagramma's van ER-varianten.

JDBC Java Database Connectivity: een groep klassen die het mogelijk maken om vanuit de programmeertaal Java een connectie te leggen met een databank en dan op die databank operaties uit te voeren.

PL/SQL Programming Language / Structured Query Language: De SQL-variant van het Oracle8 DBMS. Het belangrijkste kenmerk is dat SQL uitgebreidt wordt met procedurele aspecten zoals functies en lokale variabelen.

SCOB StudieCentrum voor Onderneming en Beurs

SQL Structured Query language: Vraagtaal voor relationele databanken. Er bestaat SQL-standaard maar er zijn ook veel uitbreidingen zoals temporele SQL, SQL ingebed in programmeerd talen (*embedded SQL*), procedurele SQL, SQL voor datamining. Elk DBMS heeft zo zijn eigen uitbreidingen.

# Referenties


- **[BOHL97]** BOHLEN M.H., SNODGRASS R.T. & SOO M.D., «*Coalescing in Temporal Databases*», 1997
- **[BUEL99]** BUELENS F., «*Koersboekstructuur van de Beurs van Brussel (1832-1999)*», RUCA, 1999
- **[BYTE9709]** MULLER R.J., «*Oracle8: Worth the Wait?*» in *BYTE*, vol. 22, nr. 9, p. 111-113, september 1997
- **[BYTE9806]** WATTERSON et al., «*Enterprise databases battle on*» in *BYTE*, vol. 22, nr.6, p. 93-105, juni 1998
- **[GAND97]** GANDHI M., ROBERTSON E.L. & GUCHT, D. Van, «*Leveled Entity Relationship Model*», Computer Science Dept., Indiana University, Bloomington,1997
- **[GEMI98]** GEMIS M., BUELENS F. e.a., «*Databank Archief van de Beurs van Brussel*», working paper, UA, 1998
- **[GREG97]** GREGERSEN H. & JENSEN C.S., «*Temporal Entity-Relationship Models – a Survey*», Denemarken, 1997
- **[GREG98]** GREGERSEN H. & JENSEN C.S., «*Conceptual Modeling of Time-Varying Information*», Denemarken, 1998
- **[INF98]** «*Informix TimeSeries DataBlade Module – User's Guide*», versie 3.1, Informix Software Inc., 1998
- **[JENS97]** JENSEN C.S. & SNODGRASS R.T.,«*Semantics of Time-Varying Attributes and Their Use for Temporal Database Design*», VS, 1997
- **[LEE98]** LEE J.Y. & ELMASRI R.A., «*An EER-based Conceptual Model and Query Language for Time-Series Data*», University of Texas, Arlington, VS, 1998
- **[SNOD97]** SNODGRASS R.T., BOHLEN M.H., JENSEN C.S. & STEINER A., «*Transitioning Temporal Support in TSQL2 to SQL3*», 1997
- **[SNOD98]** SNODGRASS R.T., «*Managing Temporal Data – A Five-Part Series*», 1998
- **[STON97]** STONEBRAKER M., «*Architectures for Object-Relational DBMSs: The good, the bad, the ugly and the non-existent*», Informix Whitepaper, 1997
- **[TORP97]** TORP K., JENSEN C.S. & BOHLEN M., «*Layered Implementation of Temporal DBMSs – Concepts and Techniques*», Denemarken, 1997
- **[UA97]** UA, «*Aanvraagdossier GOA UA – Catalogisering en digitalisering van het Archief van de Beurs van Brussel*», UA, 27 januari 1997
- **[UML97]** Relational Software en anderen, «*UML Notation Guide 1.1*»,1997
- **[WILL99]** WILLEMS H., «*Voorlopige Inventaris Knipseldossiers*», RUCA, 1999

# Bibliografie

Inleidende lectuur over Java, Oracle en SQL:


- CORRIGAN P. & GURRY M., «*ORACLE Performance Tuning*», O'Reilly & Associates, VS, 1993
  ISBN: 1-56592-048-1

- DATE C. J., «*The systems programming series: An introduction to Database Systems 6th edition*», Addison-Wesley, VS, 1995
  ISBN: 0-201-82458-2
  UIA: WIS 681.37 G DATE 95

- ELMASRI R. & NAVATHE S.B., «*Fundamentals of Database Systems*», Addison-Wesley, VS, 1994
  ISBN: 0-8053-1753-8

- GROFF J.R. & WEINBERG P. N., «*Using SQL*», McGraw-Hill, VS, 1990
  ISBN: 0-07-881524-X
  UIA: WIS 681.33 G GROF 90

- HORSTMANN C.S. & CORNELL G., «*core JAVA 1.1 – Fundamentals*», Sun Microsystems Press, VS, 1997
  ISBN: 0-13-766957-7

- HORSTMANN C.S. & CORNELL G., «*core JAVA 1.1 – Advanced Features*», Sun Microsystems Press, VS, 1997
  ISBN: 0-13-766965-8

- KOCH G. & LONEY K., «*ORACLE8: The Complete Reference*», Osborne McGraw-Hill, VS, 1997
  ISBN: 0-07-882396-X

- WEINER S.R. & ASBURY S., «*Programming with JFC*», Wiley, VS, 1998
  ISBN: 0-471-24731-6

- «*JDBC™ Guide: Getting Started*», uitreksel uit «*JDBC™ Database Access from Java™: A tutorial and Annotated Reference*», JavaSoft


Inleidende lectuur over het beursgebeuren:


- ANTOINE J. & BROQUET C. & CAPIAU-HUART M-C, «*Titres en Bourse, Traité*», De Boeck, 1987
  ISBN: 2-8041-1054-0
  RUCA: (CB) 336 M-C13.522:1 b

- EYSKENS M. e.a., «*Wegwijs GELD*», Davidsfonds, Leuven, 1990
  ISBN: 90-6152-624-8

- MESSIAEN R. & VANDERLINDEN M., «*Wegwijs in de financieel-ecomonomische bericht-geving*», Uitgeversbedrijf Tijd, Antwerpen, 1994
  ISBN: 90-71986-19-5

- STURTEWAGEN B., «*Inzicht in Beleggen – Wegwijs op de beurs*», Lannoo, Tielt, 1989
  ISBN: 90-209-1680-7

# Inhoudstafel