OO overlay op relationele systemen

Naam	:	M.J.G. Thörig
Roepnaam		Martijn
Datum	:	21-juni-2004
Opleiding	:	1 jarige Master Software Engineering
Document name	:	Scriptie_Mattic_martijn (Afstudeerscriptie Master Software Engineering)
Afstudeerdocent	:	M.G.J. van den Brand
Stagebegeleider	:	Jeanot Bijpost
Opdrachtgever	:	Mattic Software B.V. / Lectureship Software Quality HvA

Voorwoord

Voor u ligt de afstudeerscriptie van M.J.G. Thörig. Mijn afstudeerstage heeft plaats gevonden bij Mattic Software. Deze afstudeerstage is een onderdeel van de 1-jarige Master Software Engineering aan de Universiteit van Amsterdam.

Ik wil hierbij de volgende mensen bedanken:

Mattic Software/ Lectureship Software Quality (Hogeschool van Amsterdam)
Het bedrijf en de instelling waaraan ik mijn stage heb te danken.
J. Bijpost, mijn begeleider vanuit Mattic Software
M.G.J. van den Brand, mijn begeleider vanuit de Universiteit van Amsterdam

Samenvatting

Cathedron is een case tool die ontwikkeld wordt door Mattic Software. Cathedron maakt gebruik van de In Memory Database, omdat de In Memory Database sneller werkt, dan wanneer gebruik wordt gemaakt van een relationele database. De In Memory Database is een relationele database die volledig in het werkgeheugen van een PC wordt geplaatst. De opbouw van de In Memory Database gebeurt aan de hand van de onderliggende relationele database. De opdracht van deze stage was om de persistency te verbeteren van de In Memory Database. Om een voorstel te kunnen doen zijn er uiteindelijk drie oplossingsrichtingen bedacht. Het zijn oplossingsrichtingen, omdat niet een complete model wordt opgesteld om tot een oplossing van het probleem te komen. Dit komt doordat de oplossingsrichtingen meer aanpassingen nodig hebben dan in eerste instantie werd gedacht.

Hier volgt de samenvatting per hoofdstuk.

Hoofdstuk 1 is de inleiding van deze scriptie.
Hoofdstuk 2 is een inleidend verhaal over Mattic Software en haar producten. Verder wordt de achtergrond van de onderzoeksvragen toegelicht. Daarna worden de onderzoeksvragen uitgeschreven. De doelstelling van dit onderzoek luidt: "Wat is de beste manier om de persistency van de In Memory Database te verbeteren?"
Hoofdstuk 3 gaat over het opgestelde Plan van Aanpak.
Hoofdstuk 4 gaat over de concepten van een object georiënteerde database en een relationele database. Er wordt aangegeven of de vergelijking tussen een object georiënteerde programmeer taal en een database opgaat. Verder wordt aangegeven waarom de impedance mismatch betrokken wordt bij dit onderzoek. Er wordt een lijst opgesteld van die concepten om vervolgens aan te geven welke concepten niet geïmplementeerd worden in de In Memory Database. Nu kan Mattic Software aangeven welke andere niet ondersteunde concepten nog meer betrokken worden bij de oplossingsrichtingen.
Hoofdstuk 5 gaat over de drie oplossingsrichtingen. Een oplossingrichting is een niet compleet model om tot een oplossing voor het probleem te komen. Uiteindelijk wordt er een voorstel gedaan om te kiezen voor een oplossing waarvan de werking lijkt op een 'versie beheer tool' en zo min mogelijk netwerkverkeer wordt veroorzaakt (dit telt onder andere mee met de keuze die Mattic Software moet maken).
Appendix 1 is een lijst waarin bepaalde termen met hun definitie staan.
Appendix 2 is de vergelijking van een object georiënteerde programmeertaal, een object georiënteerde database en een relationele database. Het doel van deze vergelijking was om aan te geven dat de vergelijking tussen een object georiënteerde programmeertaal en database niet opgaat.
Appendix 3 is een toelichting op de Impedance Mismatch. Impedance Mismatch is een terminology die gebruikt wordt voor de mapping van twee systemen, die beide een andere conceptuele basis hebben. Hier gaat het om de mapping van een relationele database en klassen van een object georiënteerde programmeertaal.
Appendix 4 is de literatuurlijst.

Inhoudsopgave

1 Introductie

Deze scriptie wordt geschreven naar aanleiding van mijn afstudeerstage bij Mattic Software. De stage had ten doel te onderzoeken wat de beste mogelijkheid is om de In Memory Database, IMD, van Mattic Software te verbeteren, met de intentie om de IMD beter persistent te maken. De IMD is een onderdeel van Cathedron[1]. De IMD wordt gebruikt als Cache database om data van de gebruikte relationele database in het geheugen van de PC te plaatsen. De aanpak om tot een voorstel te komen bestaat uit de volgende verschillende stappen:
Het vergelijken van de concepten van de object georiënteerde database, de relationele database en de object georiënteerde programmeertaal. In de praktijk wordt vaak gesproken over 'het verschil tussen relationeel en object georiënteerde'. Door de vergelijking van de concepten willen we duidelijk maken dat de vergelijking tussen een object georiënteerde programmeertaal en een database systeem niet opgaat.
Uiteindelijk wordt ook de implementatie van de In Memory Database, IMD, van Mattic Software beoordeeld aan de hand van de concepten. Zo ontstaat een beeld van de gebreken en kan beoordeeld worden welke verbeteringen noodzakelijk zijn.
Hierna wordt aangegeven of de concepten ook zijn verwerkt in de In Memory Database. Op basis van deze stap worden er concepten betrokken bij het voorstel voor verbetering van de persistentie van de In Memory Database.
Daarna volgende de verschillende oplossingsrichtingen om de persistentie te verbeteren van de In Memory Database. Een oplossingrichting is een niet compleet model om tot een oplossing voor het probleem te komen.

[1] Zie ook het volgende hoofdstuk voor meer informatie over Cathedron.

2 Mattic Software

Mattic Software is een kleine onderneming met als core business het ontwerpen, bouwen en implementeren van informatiesystemen. De opbrengst hiervan wordt gebruikt om de ontwikkeling van de twee case-tools van Mattic Software te bekostigen. De twee case-tools zijn Infagon en Cathedron. Infagon is een case-tool voor het ontwerpen van databases en Cathedron is een case-tool voor het bouwen van informatiesystemen.

2.1 Infagon

Infagon bestaat al 10 jaar en is beter bekend onder de naam Fully Communication Oriented Information Modeling, FCO-IM, Case-tool. FCO-IM wordt op veel plaatsen in Nederland binnen het hoger onderwijs gebruikt. Het is een tool, die gebaseerd is op de FCO-IM methode[1]. Deze methode is een verbetering van de Natuurlijke taal Informatie Analyse Methode[2], NIAM methode. Wat Mattic betreft ligt de kracht van FCO-IM niet zo zeer in de taal analyse; Mattic is vooral geïnteresseerd in het feit dat FCO-IM een methode is die zo elementair is, dat daaruit eenvoudig conversies kunnen plaatsvinden naar andere veel gebruikte methoden zoals EAR, ER, UML, etc (meer informatie hierover zie www.infagon.com).

2.2 Cathedron

Cathedron bestaat ruim twee jaar en wordt tot nu toe alleen toegepast tijdens interne projecten. Informatiesystemen kunnen razend snel worden ontwikkeld via Cathedron. Ontwikkeling met Cathedron gaat zo snel, omdat er zoveel mogelijk gebruik wordt gemaakt van de structuur van de onderliggende database. Dit is een techniek/methode die bekend staat onder de naam Model-Based Application Development.
Het algemene idee bestaat al jaren, maar de verschillende tools die deze methode hanteren verschillen sterk in functionaliteit en bruikbaarheid. Zo zijn er tools die broncode genereren op basis van UML modellen en er zijn tools die vrijwel zonder broncode werken. Mattic heeft getracht met Cathedron een oplossing te bedenken die de beste zaken van verschillende ideeën combineert. Zo kan binnen Cathedron, zonder te programmeren, eenvoudige schermen worden gemaakt met één druk op de knop. Schermen met meer complexiteit kunnen door eenvoudige scripts worden gemaakt. Wanneer er meer behoefte is aan schermen met een nog grotere complexiteit, dan kan gebruik worden gemaakt van de ontwikkelomgeving Delphi van Borland. Dit is mogelijk omdat Cathedron volledig geïntegreerd kan worden binnen Delphi.

2.3 Toekomst

Mattic wil in de toekomst beide case-tools volledig gaan integeren en op een aantal cruciale punten uitbreiden. Zo moet Infagon meerdere methoden voor modellering gaan ondersteunen. Bij voorbeeld, een toevoeging van een veld op een scherm moet bij Cathedron direct aanleiding geven tot een aanpassing van de onderliggende informatiemodellen en de fysieke database.

2.4 Achtergrond en onderzoeksvragen

De beide case-tools van Mattic maken gebruik van een database in het geheugen, de zogenaamde In Memory Database, IMD. Deze database is tien jaar geleden ontworpen voor het FCO-IM Casetool, maar innovatie liet te wensen over.

Voor Infagon is de IMD de enige database. Een informatie model in Infagon wordt in de vorm van een databestand opgeslagen op de disk. Zodra het informatiemodel wordt geopend, wordt dit bestand gelezen en in de vorm van een relationele database volledig in het geheugen geplaatst.
Handelingen worden niet efficiënt door de IMD opgeslagen. Mattic Software wil deze technologie verbeteren door toepassing van een centrale rationele database die het informatiemodel bevat. De In Memory Database zou dan puur omwille van de snelheid de gegevens lokaal in het geheugen hebben en wordt daarmee dus een Cache Database.
Cathedron werkt al met een centrale relationele repository. De In Memory Database, IMD, wordt binnen Cathedron dan ook al gebruikt als een Cache Database. Probleem is echter dat de synchronisatie tussen de relationele database en de IMD volledig met de hand gecodeerd moet worden. Mattic Software wil dit probleem graag opgelost hebben. Door middel van een continue synchronisatie met de relationele database door de IMD zelf. Het werkt persistency in de hand en bevordert concurrency. Deze opdracht benadrukte persistency maar verliest het aspect van concurrency niet uit het oog.
De doelstelling van dit onderzoek is: De persistency van de IMD op een meer efficiënte manier te verbeteren! Om tot een antwoord te komen van de doelstelling, moeten vooraf een aantal vragen worden beantwoord. Het startpunt van dit onderzoek is om duidelijk te maken wat de concepten van een database zijn. Dit wordt bewerkstelligt door de analyse van de concepten van zowel de object georiënteerde database als de concepten van de relationele database. In de analyse wordt aangegeven of de concepten worden gebruikt in een object georiënteerde database, een object georiënteerde programmeertaal en een relationele database. De uitkomst kan licht werpen op de volgende uitspraak:”kan een willekeurige database gelijk worden gesteld aan een object georiënteerde programmeertaal?”
De volgende vraag die beantwoord moet worden is welke concepten van de object georiënteerde database en relationele database ondersteund worden door de In Memory Database. Dit gebeurt door een overzicht te maken van de concepten en aan te geven welke reeds worden ondersteund door de In Memory Database.
Voor deze keuze wordt een werkhypothese opgesteld. Op grond van een werkhypothese, kunnen uit de overige, besloten worden welke als eerste in aanmerking komt voor implementatie in de IMD.
De hypothese omvat de volgende punten:

1. De opsomming van object georiënteerde- en relationele databases
2. Kan een willekeurige database gelijk worden gesteld aan een object georiënteerde programmeertaal?
3. Welke concepten van de object georiënteerde database en de relationele database kunnen worden geïmplementeerd in de In Memory Database?
4. Indien juist, welke verdienen dan prioriteit bij de implementatie?
5. Wat is de meest efficiënte manier van implementeren?

Deze punten samengevat: de verbetering van de persistency in de IMD.

3 Plan van aanpak

Om de onderdeelspunten te kunnen beantwoorden, binnen de gestelde hypothese,is het plan van aanpak opgesteld in overleg met de opdrachtgever, J. Bijpost. De uitwerking van het plan van aanpak is als volgt:
De literatuurstudie zal minimaal één week in beslag nemen. Als beginpunt van de literatuurstudie is genomen het zoeken naar documentatie over object georiënteerde database en relationele database. Tijdens dit onderzoek worden de eerste en tweede stelling van de hypothese beantwoord (zie Hoofdstuk 4).
Om één van de doelen van de In Memory Database, IMD, beter te begrijpen, is een literatuurstudie gedaan naar de Impedance Mismatch. Een korte inleiding tot de Impedance Mismatch is opgenomen in Appendix 3.
Wanneer week 2 van deze stage begint, wordt er begonnen aan het analyseren van de IMD. Dit wordt gedaan door UML klassendiagrammen op te stellen van de IMD. Het klassendiagram is te vinden in Hoofdstuk 5.
De analyse van de IMD moest gedaan worden om te kunnen aangeven welke van de concepten (deze lijst is opgesteld tijdens de literatuurstudie) niet geïmplementeerd zijn in de IMD, waardoor de derde stelling van de hypothese is beantwoord (Zie Hoofdstuk 4).
Het antwoord van de vierde stelling van de hypothese wordt in Hoofdstuk 4 beschreven.
De centrale vraagstelling en de vijfde stelling van de hypothese kunnen vervolgens beantwoord worden. Hiervoor worden er één of meerdere oplossingsrichtingen gegeven. Deze oplossingsrichtingen worden voorgelegd aan Mattic Software en zij moet dan een keuze maken (zie Hoofdstuk 5).

4 Concepten van databases

Voor de concepten van de object georiënteerde database management systeem, OODBMS, is gebruik gemaakt van de publicatie van Atkinson [M. Atkinson et al] en voor de concepten van de relationele database management systeem, RDBMS, is er gebruik gemaakt van de publicatie van Codd [Codd]. Ook worden enkele concepten van een OODBMS uit de publicatie van Bertina [Elisa Bertina et al] gehaald. Er is gekozen voor de concepten van Atkinson [M. Atkinson et al], want er wordt veel naar dit document verwezen door papers die over OODBMS gaan. De publicatie van Bertina [Elisa Bertina et al] vult de lijst aan van Atkinson [M. Atkinson et al]. In het werk van Codd [Codd] wordt de basis gelegd voor de relationele database.
Er is gekozen voor de concepten van een OODBMS en relationele database, omdat de In Memory Database, IMD, is een implementatie van een relationele database in Delphi. Die op zijn beurt als taal object georiënteerd is en ontwikkeld is door Borland[1].

4.1 Doel van de lijst met concepten

Er zijn meerdere doelen waarom deze lijst met concepten wordt gemaakt. Deze zijn om:

- Aan te geven welke concepten verwerkt zijn in een relationele database als een object georiënteerde database.
- Aan te geven of een object georiënteerde programmeertaal kan worden vergeleken met een relationele database of een object georiënteerde database, zie Appendix 2.
- Aan te gegeven welke concepten verwerkt zijn in de IMD.

4.2 Concepten van OODBMS

In Atkinson [M. Atkinson et al] wordt er een definitie gegeven van een object georiënteerde database management systeem, OODBMS. Deze definitie geeft volgens Atkinson [M. Atkinson et al] de benodigde features van een OODBMS. Uit deze features worden concepten samengesteld. In Bertina [Elisa Bertina et al] staan nog meer features en concepten van een OODBMS. Met deze opsomming wordt de lijst, opgesteld door Atkinson [M. Atkinson et al], aangevuld.

- Complex Object
- Object Identity
- Encapsulation
- Information Hiding
- Types
- Hiërarchie
- Multiple inheritance
- Overriding
- Overloading
- Late binding
- Ad Hoc Query
- Computational Completeness
- Extensibility
- Schema evolution (extensibility)
- Persistency
- Secendary storage management
- Concurrency
    - Transactions
    - Locking

[1] http://www.borland.nl/delphi_net/index.html

    - Distribution
        - Replication
        - Multi User
        - Synchronisation
- Recovery
- Versioning

4.3 Concepten van de relationele database

Naast de concepten van de OODBMS worden de concepten van de relationele database gebruikt om de lijst van concepten aan te vullen. De volgende concepten worden genoemd in Codd [Codd]:

- Relation
- Relationships
- Set
- Tuple
- Row
- Column
- Domein
- Component
- Role name
- Active domein
- Primary key
- Foreign key
- Nonsimple Domein

4.4 De analyse

De analyse van de toepasbaarheid tussen databases en object georiënteerde talen.
Om de toepasbaarheid te kunnen testen wordt gebruik gemaakt van de object georiënteerde taal C++. De informatie staat in het boek van Laan [Laan]. De keuze om een object georiënteerde taal toe te passen wordt gedaan op de hypothese dat een database en een object georiënteerde programmeertaal compactible zijn. Deze analyse kan uitsluitsel geven over de toepasbaarheid van de hypothese.De informatie over het object georiënteerde managementsysteem wordt gehaald uit de documentatie over de OODBMSO2. Het systeem O2 is ontwikkeld tijdens een vijf-jarig project, genaamd Altaïr [O DUX at al]. De keuze voor O2 is gemaakt omdat het systeem IDBMSOD verregaand compleet is. Ook voldoet O2 aan de meeste eisen die in Atkinson [M. Atkinson at al.] worden gesteld. De informatie over relationele database systeem komt uit Codd [Codd]. Alle relationele database systemen zijn op basis van het beschreven relationele model, door Codd [Codd] opgesteld. Dit werkmodel is door anderen aangevuld en uitgebreid. Voor de uitwerking, zie Appendix 2.

4.4.1 Conclusie

Er zijn te weinig overeenkomsten tussen een OOPL en een database. De analyse geeft aan dat een OOPL een general purpose programming language is. Dit heeft tot gevolg dat alles opnieuw geprogrammeerd moet worden, om de implementatie toepasbaar te maken. Hierdoor kan worden gezegd dat alle genoemde concepten te maken zijn met een OOPL. De concepten zijn bruikbaar te maken door toepassing van een library.

Een standaard database systeem heeft een constrain met betrekking tot zijn functionaliteit en daardoor voldoet het niet meer aan de eigenschap general purpose. Een database heeft als doel gegevens te verwerken.
Op grond van de O2 documentatie[1] en de gemaakt analyse, kan worden geconcludeerd dat een object georiënteerde database met een object georiënteerde database management systeem meer is dan alleen een object georiënteerde taal met een library voor persistency. Op grond hiervan wijzen we stelling nummer twee van de hypothese af. Hieruit kan worden geconcludeerd dat een relationele database niet hetzelfde is als een object georiënteerde programmeertaal. De basis voor het relationele model is opgesteld door Codd [Codd] en door anderen overgenomen. De ontwikkeling heeft zich anders voltrokken bij de object georiënteerde databases, want deze zijn ontstaan door de vraag naar de mogelijkheid om complexe objecten in op te slaan; [M. Atkinson et al]. Atkinson zegt daar het volgende over: "*Whereas Codd's original paper [Codd 70] gave a clear speciation of a relational database system (data model and query language), no such speciation exists for object-oriented database systems [Maier 89]. ... Opinion is slowly converging on the gross characteristics of a family of object-oriented systems, but, at present, there is no clear consensus on what an object-oriented system is, let alone an object-oriented database system.*"[M. Atkinson et al]. Hieruit blijkt dat er geen duidelijke specificatie bestaat over de features van een OODMS op het moment dat M. Atkinson zijn publicatie schreef. Daarom wordt zijn document geciteerd met betrekking tot publicaties over OODBMS.
De vergelijking tussen een relationele database met een object georiënteerde database ligt meer voor de hand, omdat beide veelal dezelfde concepten ondersteunen. Beide systemen hebben ook hetzelfde doel: het verwerken van data. Het grote verschil tussen de twee databasesystemen is het onderliggende schema dat wordt ondersteund. Een relationele database ondersteunt het relationele schema en een object georiënteerde database ondersteunt het object georiënteerde schema. Op grond van Codd [Codd] bevindingen is elk relationeel systeem niet op dezelfde wijze geïmplementeerd. Daarnaast kan de uitspraak van Atkinson [M. Atkinson et al] dat een relationeel systeem geen complexe objecten ondersteunt, als min of meer achterhaald worden beschouwd. Dit komt doordat de relationele systemen door innovatie verbeterd zijn en het werk van Codd [Codd] op dit punt eveneens als achterhaald kan worden beschouwd. Omdat zijn werk niet meer voldoet aan de eisen van de huidige tijd. Een voorbeeld van innovatie is het Oracle Object [2]. Ondanks die ontwikkeling in de relationele database wereld, heeft Codd **[**Codd] de basis gelegd voor het relationeel systeem.

4.5 Impedance Mismatch

De noodzaak om een discussie te voeren over de Impedance Mismatch wordt ingegeven door de verwarring die ontstond door de opbouw van de In Memory Database, IMD, aan de hand van de centrale relationele database. De verwarring ontstond omdat de IMD geïmplementeerd is in Delphi, een object georiënteerde programmeertaal. Hierdoor zou het probleem met de naam Impedance Mismatch kunnen ontstaan. De Impedance Mismatch kan ontstaan wanneer er een mapping plaats vindt van twee concepten. Daarentegen is de IMD gemaakt op het relationele concept van Codd, waardoor de mapping van de IMD met een relationele database geen probleem oplevert. Daarbij worden de gebruikte klassen binnen Cathedron gegenereerd met als basis de IMD. Hierdoor onstaat er een mapping waarbij het object georiënteerd model, de genereerde klassen, en het relationele model, de IMD, betrokken zijn. Daarom treedt de Impedance Mismatch hier niet op. Zie hiervoor Appendix 3.

4.6 De verwerkte concepten van de In Memory Database

Hieronder zal worden aangegeven welke van de hier eerder opgesomde concepten zijn geïmplementeerd in de In Memory Database, IMD. Door dit aan te gegeven wordt het

[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/
[2] http://www.cise.ufl.edu/help/database/oracle-docs/java.920/a96654/oraoot.htm

overzichtelijk wat de mogelijkheden van de IMD zijn. Hierdoor wordt een antwoord gegeven op stelling drie van de hypothese. Nadat deze lijst is opgemaakt kan worden bepaald welk concept of welke concepten er aan de IMD worden toegevoegd. Hiermee wordt het vierde stelling van de hypothese beantwoord. De uitleg van de concepten staat in Appendix 2.

4.6.1 De concepten

Concept:	Complex Object		
IMD:	Nee	**Hoe:**	De In Memory database kan dezelfde objecten maken als beschreven staat in het relationele model van Codd [Codd].
Note:			

Concept:	Object Identity		
IMD:	Ja	**Hoe:**	De object indentity is hetzelfde als die in een relationele database.
Note:	Via de klasse TcmTable kan worden gezocht aan de hand van de Primary key naar TcmRecords die bij de TcmTable horen.		

Concept:	Encapsulation/ Information Hiding		
IMD:	Ja	**Hoe:**	Een tabel is geïmplementeerd als een klasse. Deze klasse gedraagt zich zoals iedere andere klasse en ondersteunt dus Information Hiding.
Note:			

Concept:	Types		
IMD:	Ja	**Hoe:**	Dat zijn de klassen die op basis van de tabellen van de In Memory database gemaakt zijn.
Note:			

Concept:	Hiërarchie		
IMD:	Ja	**Hoe:**	Het type hiërarchie wordt ondersteund door de klassen die op basis van de tabellen van de In Memory database zijn gegenereerd. Ook wordt hiërarchie ondersteund door de tabellen van de In Memory database zelf.
Note:			

Concept:	Multiple inheritance		
IMD:	Nee	**Hoe:**	Dit wordt niet ondersteund door Delphi. Mattic zelf is ook niet van plan om Multiple inheritance te ondersteunen.
Note:			

Concept:	Overriding/ Overloading/ Late binding		
IMD:	Ja	**Hoe:**	Wordt ondersteund door de klassen die op basis van de tabellen van de In Memory database gemaakt zijn.
Note:			

Concept:	Ad Hoc Query		
IMD:	Nee	**Hoe:**	Wordt niet ondersteund door de In Memory Database. De ondersteuning van SQL kan worden geïmplementeerd door gebruik te maken van een SQL interpreter en deze te koppelen aan de In Memory

			database.
Note:			

Concept:	Computational Completeness		
IMD:	Ja	**Hoe:**	Het model van de Relationele database is computational compleet.
Note:			

Concept:	Extensibility		
IMD:	Ja	**Hoe:**	De gegenereerde klassen kunnen worden aangepast door de structuur van de tabellen aan te passen.
Note:			

Concept:	Schema evolution		
IMD:	Ja	**Hoe:**	Door het schema van de tabellen aan te passen in de IMD.
Note:	TcmTable kan worden aangepast door bijvoorbeeld een TcmColumn toe te voegen.		

Concept:	Persistency		
IMD:	Ja	**Hoe:**	Er wordt gebruik gemaakt van serialisation. Bij dit proces worden alle pointers vervangen door gegenereerde nummers en wordt alle data sequentieel in een data bestand geschreven. Dit moet worden uitgebreid met ondersteuning voor de relationele database.
Note:	De IMD wordt nu in zijn geheel gekopieerd naar een relationele database.		

Concept:	Secondary storage management		
IMD:	Nee	**Hoe:**	Is niet nodig, maar de mogelijkheid om meerdere database aan te spreken bestaat. Het wordt mogelijk om een andere database als secondary storage te gebruiken.
Note:			

Concept:	Concurrency Transactions Locking Distribution Replication Multi User Synchronisation		
IMD:	Nee	**Hoe:**	Wordt nu niet ondersteund, maar zal wel terug komen in de uitkomsten.
Note:			

Concept:	Recovery		
IMD:	Nee	**Hoe:**	Wordt nu niet ondersteund, maar zal wel terug komen in de uitkomsten.
Note:			

Concept:	Version		
IMD:	Nee	**Hoe:**	Version wordt niet ondersteund door de IMD.
Note:			

Concept:	Relation/ Relationships

IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:			

Concept:	Set		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:	Er is ook een klasse TcmSet.		

Concept:	Tuple		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:	De klasse TcmRecord kan worden vergeleken met een Tuple.		

Concept:	Row		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:	De weergave van een klasse TcmRecord kan worden vergeleken met een Row.		

Concept:	Column		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:	De klasse TcmColumn is hetzelfde als een Column.		

Concept:	Domein		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:	De klasse TcmColumn kent de fIdentifier: TmName; object dat gelijk is aan het domein.		

Concept:	Component		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:	Hiervoor moeten de columns van een table worden getoond.		

Concept:	Role name		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:	Ja, dat kan door gebruik te maken van de pointer verwijzing.		

Concept:	Active domein		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen.
Note:			

Concept:	Foreign key/ Nonsimple Domein		
IMD:	Ja	**Hoe:**	De IDM is een relationele database, maar dan in het werkgeheugen, zie ook object identity.
Note:			

4.7 Conclusie

De In Memory Database, IMD, bevat veel van de genoemde concepten. Hierdoor kunnen wij modellen opstellen om tot een oplossing te komen "oplossingsrichtingen". In overleg met Mattic Software is er voor gekozen om concurrency en recovery bij de modellen te betrekken. En vanzelf sprekend persistency. Tijdens dit overleg kwam naar voren dat als het gaat om concurrency, latency en bandbreedte ook meetellen met de keuze van Mattic. Met latency wordt bedoeld, de vertraging die wordt opgelopen door packets tijdens het versturen van de client naar de server. Met bandbreedte wordt bedoeld de grote van een packet. Dit komt doordat Cathedron een client programma is die gebruik maakt van een centrale relationele repository. De centrale relationele repository is dan de server side. Hierdoor moet er rekening worden gehouden met het netwerkverkeer tussen Cathedron en de centrale relationele repository. Hiermee is stelling vier van de hypothese beantwoord.

5 De oplossingsrichtingen

Op grond van de voorafgaande conclusie wordt in dit hoofdstuk drie oplossingsrichtingen besproken.

5.1 Eerste oplossingsrichting

Deze oplossingsrichting maakt een directe vertaalslag tussen alle verricht handelingen en SQL. Dit wordt vervolgens opgestuurd naar de centrale repository, waardoor de persistency wordt bevorderd.

5.1.1 Persistency

Huidige inrichting

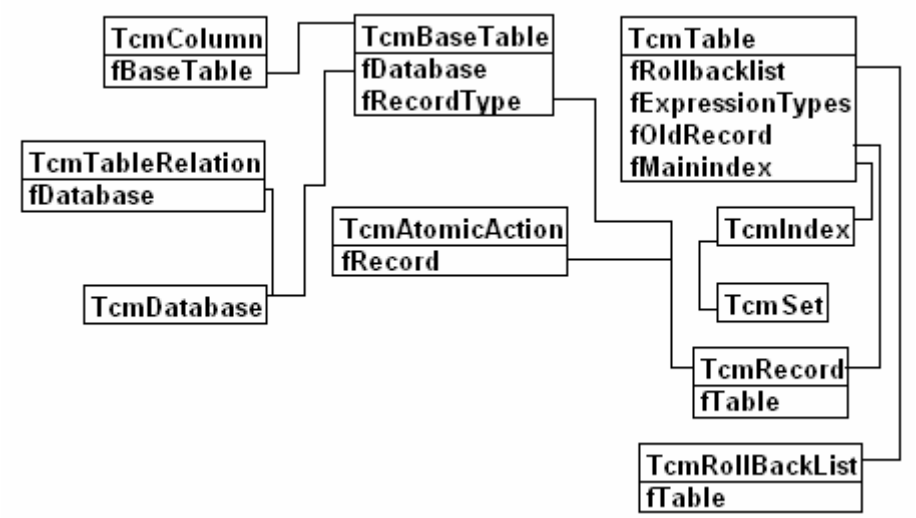
Door huidige werking van de In Memory Database, IMD, kan die niet worden toegepast in de oplossingsrichtingen. Om de persistency te verbeteren, moet onder andere de rollbacklist worden aangepast. De rollbacklist wordt gebruikt om alle handelingen die verricht worden op de IMD bij te houden. Wanneer deze handelingen ongedaan moeten worden gemaakt, gebeurt dit via de rollbacklist. Om dit te implementeren in de IMD is er een klasse gemaakt met de naam rollbacklist. Deze klasse heeft een stack waarin alle handelingen worden geplaatst die verricht worden op een tabel. Worden er handelingen ongedaan gemaakt, dan worden één voor één de handelingen van de top van de stack gehaald. De handelingen worden verwijderd uit de stack wanneer ze definitief worden vastgelegd via de commithandeling.
De huidige rollbacklist zal voor de oplossingrichtingen aangepast moeten worden. Er moet één rollbacklist object zijn voor de hele database en deze houdt alle handelingen die verricht worden op de database bij. In de huidige IMD is één rollbacklist object gekoppeld aan één tabel object, zie figuur 1.

Aanpassingen

Om de persistency te verbeteren worden alle acties op de In Memory Database, IMD, opgeslagen in een extra stack. Deze stack wordt een onderdeel van de rollbacklist. In de nieuwe situatie moet, wanneer een commit wordt gedaan, SQL statements worden gegenereerd. Vervolgens worden alle gegenereerde SQL statements verstuurd naar de centrale relationele database. Wanneer de centrale relationele database aangeeft dat de veranderingen doorgevoerd kunnen worden, wordt de stack van rollbacklist van de IMD leeggemaakt. Nu zijn alle handelingen permanent. De centrale relationele database kan ook een foutmelding geven en dan moet de rollback worden uitgevoerd op de IMD.
Er zijn verschillende handelingen die verricht kunnen worden op een relationele database. De volgende handelingen zijn van belang voor dit onderzoek: update, add en delete. De IMD ondersteunt ook deze handelingen, maar de update is anders geïmplementeerd. Wanneer er een update plaats vindt in de IMD, dan wordt de betrokken relatie eerst verwijderd via de delete en weer teruggezet in zijn nieuwe vorm via de add. Echter, tijdens deze handeling blijft de references naar de aangepaste relatie intact. De rollbacklist krijgt de update binnen als een delete met daarop een add. Wanneer dit zou worden omgezet naar SQL, dan zou in een relationele database een compleet nieuwe relatie worden aangemaakt. De references naar de verwijderde relatie zouden op dat moment de waarde NULL krijgen. Dit probleem kan worden voorkomen door de rollbacklist aan te passen. De rollbacklist heeft een methode waarmee alle handelingen worden aangemeld bij de rollbacklist. Door de parameters van deze methode uit te breiden met een extra Boolean, kan worden aangegeven of de handeling een update, add of delete is. Daarnaast moet de klasse TcmAtomicAction worden

[1] http://www.cise.ufl.edu/help/database/oracle-docs/appdev.920/a97269/pc_03dbc.htm

aangepast. Deze klasse wordt gebruikt om de verschillende handelingen op de IMD in de stack van de rollbacklist op te slaan. De TcmAtomicAction moet ook aan kunnen geven of het gaat om een update.



Figuur 1

5.1.2 Concurrency

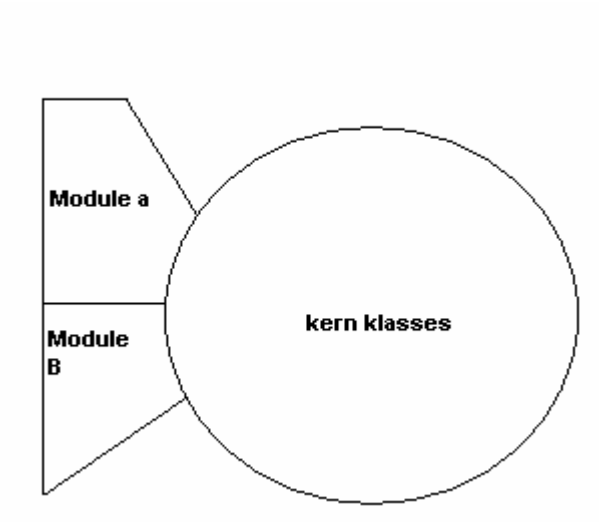
Cathedron is een client applicatie, wat al eerder is toegelicht. Deze applicatie maakt gebruik van de In Memory Database, IMD. Deze eerste oplossingsrichting lijkt binnen een multi user omgeving een verkeerde keuze, omdat de foutmelding van de relationele database pas komt wanneer de commit wordt uitgevoerd op de IMD. Echter, er is een kleine kans op een foutmelding. Dit komt door de visie op softwareontwikkeling van Mattic Software. Deze visie wordt hieronder verder toegelicht.

5.1.3 Mattic's visie op ontwikkelen

Mattic heeft een eigen visie ontwikkeld op het ontwikkelen van software, maar ook hoe een case-tool zou moeten werken. Mattic ziet de ontwikkeling van software als volgt. Mattic begint met het ontwikkelen van de database. Dit is een relationele database. Op basis van de structuur van de database worden de benodigde klassen gegenereerd. De data van deze klassen zijn dus persistent doordat ze gegenereerd zijn door Cathedron. Een ander voordeel van het genereren van deze klassen is dat de impedance mismatch is voorkomen, zie Appendix 3 voor meer informatie over de impedance mismatch.
De gegenereerde klassen zijn de kern voor het te ontwikkelende programma, figuur 2. Voorbeeld van kernklassen kunnen bij een relatie management systeem, RMS zijn: persoon, klant, bedrijf, factuur, factuurregel. Omdat deze klassen de basis zijn voor het hele programma, worden deze klassen vooraf door het team ontworpen. Tijdens het ontwerpen wordt nog niet gewerkt aan de overige delen van het systeem.
Nadat de kernklassen gemaakt zijn kunnen deze worden uitgebreid met extra klassen voor specifieke functionaliteit. De kernklassen worden opnieuw gebruikt via extensibility. Wanneer er meerdere aparte functionaliteiten worden toegevoegd, dan hebben die tot gevolg dat beide functionaliteiten in aparte modules/libraries kunnen worden geplaatst. Er kunnen twee ontwikkelaars één van deze functionaliteiten toevoegen, figuur 2. Echter, wanneer er iets aan de kernklassen moet worden aangepast, dan stopt de verdere ontwikkeling van de modules. Dit komt ook omdat het relationele schema binnen de centrale relationele database moet worden aangepast, wanneer een aanpassing plaats vindt aan de kernklassen.

[1] http://www.cise.ufl.edu/help/database/oracle-docs/appdev.920/a97269/pc_03dbc.htm

Een voorbeeld van deze manier van ontwikkelen is als volgt: Er zijn drie ontwikkelaars. Deze ontwikkelaars moeten een Customer Relationship Manager tool, CRM, maken. De drie ontwikkelaars overleggen over de basis structuur van het programma. Dit bestaat uit een tabel voor de klant met zijn contactgegevens. Dit vormt op dit moment de kernklasse van de CRM tool. Wanneer de uitbreidingen financiën en logistiek worden ontworpen komen de ontwikkelaars erachter dat de bestaande kern niet voldoet aan de eisen van de uitbreidingen. Er moeten namelijk meerdere adressen worden gekoppeld aan de klant. Hierdoor moet de kern worden aangepast en een vertraging in de hand werkt van de verdere ontwikkeling van de uitbreiding financiën en logistiek. Wanneer de kern is aangepast kunnen de uitbreidingen op hetzelfde moment worden ontwikkeld. De ontwikkelaars van beide uitbreidingen hoeven elkaar niet in de weg te zitten. Wanneer er een verandering wordt gemaakt in de uitbreiding financiën heeft dat geen gevolg voor de uitbreiding logistiek.
Iedere ontwikkelaar kan een deel van het werk aannemen zonder dat hij een ander tot last is, dit wordt door Mattic vrijwillige locking genoemd. Door vrijwillige locking komt het bijna niet voor dat de relationele database een foutmelding teruggeeft wanneer er een commit wordt gedaan.
Omdat er geen gebruik wordt gemaakt van de two-phase commit, krijgt een ontwikkelaar niet de veranderingen door van de centrale relationele database wanneer een andere ontwikkelaar zijn veranderingen doorgeeft. Omdat een two-phase commit, veranderingen die gemaakt worden door een client verstuurd naar alle andere clients.



Figuur 2

5.2 Tweede oplossingsrichting

Deze oplossingsrichting betrekt de transactiemogelijkheid van de relationele database. Een relationele database kan meerdere clients op hetzelfde moment afhandelen. Een client kan zijn handelingen versturen naar de database. Om dit te doen, moet de client een transactie beginnen. Tijdens de transactie worden alle handelingen verstuurd naar de server.

5.2.1 Persistency

Voor deze oplossingsrichting geldt dezelfde verandering als genoemd bij de eerste oplossingsrichting als het gaat om de rollbacklist en de handeling update. Het verschil met de eerste oplossingsrichting is dat de SQL nu direct voor iedere handeling wordt gegenereerd en verstuurd naar de centrale relationele database. Dit heeft tot gevolg, dat wanneer er een transactie wordt begonnen op de IMD, er ook meteen een transactie

[1] http://www.cise.ufl.edu/help/database/oracle-docs/appdev.920/a97269/pc_03dbc.htm

gestart wordt op de centrale relationele database. Elke handeling op de IMD wordt meteen doorgestuurd naar de centrale relationele database. Hierdoor wordt het eeder bekend of de handeling ook verricht kan worden op de centrale relationele database. Dit heeft tot gevolg dat een eventuele fout sneller aan de client wordt gemeld.

5.2.2 Concurrency

Ook bij deze oplossingsrichting wordt gebruik gemaakt van vrijwillige locking, zoals bij oplossingsrichting 1 is uitgelegd.

5.3 Derde oplossingsrichting

De laatste oplossingsrichting is bijna hetzelfde als de eerste oplossingsrichting. Maar, de locking van het deel waaraan de ontwikkelaar werkt, wordt anders geregeld. Het mechanisme voor deze locking wordt door Mattic Software 'Logical' locking genoemd.

5.3.1 Locking

Locking kan op verschillende niveau's binnen een database. Dit kan zijn op tabel niveau of column niveau[1]. Locking heeft de functie de integriteit van de gegevens binnen de database te bewaken. Om de integriteit te bewaken van gegevens, worden delen van een database alleen beschikbaar gesteld voor een bepaalde gebruiker. Dit vindt plaats wanneer er een transactie wordt gestart tot het moment dat de commit is verwerkt, dan wordt dat deel weer vrij gegeven voor andere gebruikers.
Het kan ook voorkomen dat twee gebruikers toch hetzelfde deel van de database aanpassen. In een dergelijke situatie ontstaat er een deadlock. Hoe een dergelijke situatie wordt opgelost, is afhankelijk van de ontwikkelaar van de database.

5.3.2 Logical Locking

Logical Locking heeft veel weg van een versie beheer systeem, VBS, voor bijvoorbeeld de broncode. Met Cathedron worden schermen ontwikkeld. Schermen tonen gegevens van één of meerdere tabellen. De ontwikkelaar geeft dan aan bij de database dat het scherm aangepast gaat worden. Daarbij worden alleen de gegevens gelockt die op dat scherm getoond worden. Als het scherm aangepast is en dit via een commit aan de centrale relationele database wordt doorgegeven, worden de bijbehoorde gegevens ook vrijgegeven voor andere ontwikkelaars.
Wanneer een ontwikkelaar zonder aan te geven een scherm begint aan te passen, dan kan de database deze aanpassingen weigeren. Daarom is de ontwikkelaar verplicht eerst het scherm te 'locken' en nadat de aanpassingen zijn gedaan vrij te gegeven.
Er is een verschil tussen een dergelijke versie beheer via een relationele database en een gewone VBS. Er kan nooit meer dan één versie van een scherm aanwezig in een relationele database. Een VBS houdt alle veranderingen juist bij. Hierdoor kan een ontwikkelaar altijd terug naar een vorige versie van een programma, als de VBS de broncode van een programma beheert.

5.4 Voor- en nadelen

Hier worden kort de voor- en nadelen van de oplossingsrichten genoemd.

[1] http://www.cise.ufl.edu/help/database/oracle-docs/appdev.920/a97269/pc_03dbc.htm

5.4.1 Latency en brandbreedte

Alle drie oplossingsrichtingen hebben zowel voor- als nadelen. Hieronder staat een overzicht met voor- en nadelen met betrekking tot latency en bandbreedte. Met latency wordt bedoeld de vertraging die wordt opgelopen door packets tijdens het versturen van de client naar de server. Met bandbreedte wordt bedoeld de grote van een packet.

Oplossingsrichting	Latency	Bandbreedte
1	Weinig, want alleen tijdens een commit, wordt er één packet of meerdere verstuurd. Dit is afhankelijk van het aantal SQL statements dat verstuurd moet worden.	Een packet kan tot de ingestelde maximum grootte groeien. Als een packet die overschrijdt, dan moet er logischer wijze meerdere packets worden verstuurd.
2	Veel, want elke handeling wordt verstuurd naar de server. Dit heeft hetzelfde effect als de SQL meteen wordt verstuurd naar de server zonder dat er gebruik wordt gemaakt van de In Memory Database. Dit gaat ten koste van de snelheid, waardoor de Cache database geen nut meer heeft.	Een packet bevat maar één handeling, waardoor de packets klein blijven.
3	Meer latency dan bij oplossingsrichting 1, maar dit komt doordat het scherm eerst ge-lockt moet worden. Tijdens de commit wordt er één packet met alle SQL statements verstuurd naar de server.	Wanneer een scherm moet worden ge-lockt hoeft het packet niet groot te zijn; echter, wanneer de commit wordt gedaan, zal het packet groter zijn. Dit komt doordat alle handelingen erin vermeld staan. Wanneer een packet over de ingestelde maximum grootte gaat, wordt de inhoud van het packet verdeeld over meerdere packets.

5.4.2 Recovery

De recovery is voor alle oplossingsrichtingen hetzelfde. Recovery van gegevens van de client kan alleen van gegevens die op de centrale relationele database staan, de laatste aanpassingen door een commit. Wanneer een fout optreedt tijdens een transactie, dan zijn die handelingen verloren. Waardoor alle handelingen van die transactie opnieuw moeten worden doorgevoerd.

5.5 Conclusie en aanbevelingen

De persistency wordt door elke oplossingsrichting ongeveer op dezelfde manier opgelost. Omdat de In Memory Database staat in contact met een centrale relationele database. Alle handelingen worden doorgegeven aan deze centrale relationele database door de IMD. Het moment van doorsturen van de handelingen verschilt voor oplossingsrichting 2 van de andere twee.
Mattic Software heeft al min of meer aangegeven naar welke oplossingsrichting hun voorkeur uitgaat. Mattic Software heeft als voorkeur een oplossing dat zo min mogelijk

latency veroorzaakt op het netwerk. Daarnaast is er een verlies van snelheid, doordat elke SQL statement door de server moet worden gecontroleerd, gevolgd door een bevestiging. Hierdoor valt oplossingsrichting 2 af.
De aanbeveling voor Mattic Software is voor oplossingsrichting 3 te kiezen, door middel van een stapsgewijze implementatie. Voor mij is het nu niet bekend wat de repository kan als het gaat om locking, laat staan 'logical' locking. Dit komt doordat de code van de repository nog niet is geanalyseerd. Derhalve kan ik geen uitspraak doen over de repository in samenhang met locking of logical locking.
Omdat oplossingsrichting 1 veel lijkt op oplossingsrichting 3, verdient de aanpak van deze oplossingsrichting dat eerst oplossingrichting 1 wordt geïmplementeerd. Bovendien ga ik ervan uit dat bij de aanpak van oplossingsrichting 3 een protocol ontworpen moet worden. Via dit protocol wordt aangegeven welke gegevens van tabellen door de ontwikkelaar worden gebruikt bij het aanpassen van een scherm. Ook wordt doorgegeven welke gegevens in de database erbij komen wanneer de ontwikkelaar een nieuw scherm ontwikkelt. Wanneer de transactie klaar is, kan via het protocol alle gegevens die gelockt zijn vrijgegeven worden. Is het protocol voltooid, dan kan de repository worden aangepast in Cathedron.
Ik stel voor om oplossingsrichting 3 daadwerkelijk toe te passen en niet oplossingsrichting 1, omdat een ontwikkelaar bij het locken van een scherm al de melding kan krijgen dat hij deze handeling niet mag uitvoeren. Bij oplossingsrichting 1 wordt de ontwikkelaar pas tijdens de commit erop gewezen dat zijn aanpassingen niet mogelijk zijn. Dit laatste kan voor veel ergernis zorgen. Wanneer een ontwikkelaar zich begint te ergeren aan een tool, dan is de tool geen lang leven beschoren. Echter, dit speelt pas een rol wanneer Cathedron een commercieel product gaat worden.
Logical locking past ook goed bij de visie van Mattic Software op ontwikkeling van software. Logical locking vermindert de kans op een eventuele deadlock, dan wanneer er alleen gebruik wordt gemaakt van vrijwillige locking.

6 Zelf assessment

1. Kwaliteit van het onderzoeksresultaat
8, want met meer tijd kan er een beter onderzoek worden gedaan naar een totale oplossing. Daarbij kan deze oplossing kan dan getoetst worden in de praktijk.

2. Kwaliteit van de scriptie
9, want ik heb er nu veel tijd ingestoken.

3. Moeilijkheidsgraad van de onderzoeksvraag
7, want de opdracht duidelijk formuleren was een moeilijker. Alle informatie die nodig was om de doelstelling van het onderzoek te bereiken was voorhanden.

4. Relevantie van de vakken uit de Master Software Engineering voor de uitvoering van dit project
6, want er zijn te weinig raakvlakken met de gegeven vakken. Er is gesproken over de OO programmeertaal, maar het onderwerp databases is niet zo behandeld tijdens de lessen, dat ik er ook daadwerkelijk iets aan had.

Appendix 1. - Woordenlijst

Hier volgt een lijst met definities van veel gebruikte woorden. De definities zijn niet vertaald om zo te voorkomen dat er een fout in de definitie komt.

Woord	Definitie	Bron
database	n : an organized body of related information	http://dict.die.net/database/ Source: WordNet (r) 1.7
Note		

Woord	Definitie	Bron
database	1. One or more large structured sets of persistent data, usually associated with software to update and query the data. A simple database might be a single file containing many records, each of which contains the same set of fields where each field is a certain fixed width. A database is one component of a database management system.	http://dict.die.net/database/ Source: The Free On-line Dictionary of Computing (09 FEB 02)
Note		

Woord	Definitie	Bron
database management system	n : a software system that facilitates the creation and maintenance and use of an electronic database [syn: DBMS]	http://dict.die.net/database/ Source: WordNet (r) 1.7
Note		

Woord	Definitie	Bron
relational database	n : a database in which relations between information items are explicitly specified as accessible attributes; "in a relational database the data are organized as a number of differently sized tables"	http://dict.die.net/database/ Source: WordNet (r) 1.7
Note		

Woord	Definitie	Bron
Object-oriented database	n : a database in which the operations carried out on information items (data objects) are considered part of their definition	http://dict.die.net/database/ Source: WordNet (r) 1.7
Note		

Woord	Definitie	Bron
Impedance mismatch	In programming terminology it refers to the attempt to connect two systems that have very different conceptual bases, most commonly when trying to use a SQL database from an object oriented program.	http://www.free-definition.com/Impedance-mismatch.html
Note	Er is een tweede betekenis, deze is: It is a problem in electrical engineering that occurs when two transmission lines or circuits with different impedances are connected. This can cause signal reflection resulting in attenuation and noise. http://www.free-definition.com/Impedance-mismatch.html	

Woord	Definitie	Bron
Transactio	A unit of interaction with a DBMS or similar system.	http://dict.die.net/tra

n	It must be treated in a coherent and reliable way independent of other transactions.	nsaction/ Source: The Free On-line Dictionary of Computing (09 FEB 02)
Note		

Woord	**Definitie**	**Bron**
Atomic	2. [primarily techspeak] Guaranteed to complete successfully or not at all, usu. refers to database transactions. If an error prevents a partially-performed transaction from proceeding to completion, it must be "backed out," as the database must not be left in an inconsistent state.	http://dict.die.net/atomic/ Source: Jargon File (4.3.1, 29 Jun 2001)
Note		

Woord	**Definitie**	**Bron**
Two-phase commit	A feature of transaction processing systems that enables databases to be returned to the pre-transaction state if some error condition occurs. A single transaction can update many different databases. The two-phase commit strategy is designed to ensure that either all the databases are updated or none of them, so that the databases remain synchronized.	http://www.weboped ia.com/TERM/T/two_ phase_commit.html
Note		

Woord	**Definitie**	**Bron**
rollback	<database> Reverting data in a database to an earlier state, usually in response to an error or aborted operation. In a transaction based database system, transactions are considered atomic. If an error occurs while performing a transaction, the database is automatically rolled back to the state at the previous commit. Rollback may also be performed by an explicit rollback transaction.	http://dict.die.net/rollback/ Source: The Free On-line Dictionary of Computing (09 FEB 02)
Note		

Woord	**Definitie**	**Bron**
deadlock	<parallel, programming> A situation where two or more processes are unable to proceed because each is waiting for one of the others to do something... Another example, common in database programming, is two processes that are sharing some resource (e.g. read access to a table) but then both decide to wait for exclusive (e.g. write) access.	http://dict.die.net/deadlock/ Source: The Free On-line Dictionary of Computing (09 FEB 02)
Note		

Woord	**Definitie**	**Bron**
repository	1. <database> See data dictionary. 2. The core of a CASE tool, typically a DBMS where all development documents are stored.	http://dict.die.net/repository/ Source: The Free On-line Dictionary of Computing (09 FEB 02)

Note	

Woord	Definitie	Bron
data dictionary	<database> A data structure that stores meta-data, i.e. data about data. The term "data dictionary" has several uses. Most generally it is a set of data descriptions that can be shared by several applications. Usually it means a table in a database that stores the names, field types, length, and other characteristics of the fields in the database tables. An active data dictionary is automatically updated as changes occur in the database. A passive data dictionary must be manually updated. In a DBMS, this functionality is performed by the system catalog. The data dictionary is a more general software utility used by designers, users, and administrators for information resource management. The data dictionary may maintain information on system hardware, software, documentation, users, and other aspects. Data dictionaries are also used to document the database design process itself and can accumulate meta-data ready to feed into the system catalog.	http://dict.die.net/data dictionary/ Source: The Free On-line Dictionary of Computing (09 FEB 02)
Note		

Woord	Definitie	Bron
packet	n 1: a collection of things wrapped or boxed together [syn: package, bundle, parcel] 2: (computer science) a message or message fragment 3: a small package or bundle 4: a boat for carrying mail [syn: mailboat, mail boat, packet boat]	http://dict.die.net/packet/ Source: WordNet (r) 1.7
Note		

Appendix 2. - Analyse

Concept:	Complex Objects	
Beschrijving:	Er is een onderscheid te maken tussen complexe objecten en atomaire objecten. Atomaire objecten zijn bijvoorbeeld: integer, float en boolean. Een complex object is een samenstelling verschillende atomaire objecten en gedrag. Gedrag ontstaat door functies en methodes. Voorbeelden van complexe objecten zijn: bag, tuple, set, list en een array.	
OODBMS	**OOPL**	**RDBS**
O2 kan omgaan met klassen geschreven in onder andere Java en C++. De genoemde complex objecten ondersteunt O2[1].	Een OOPL ondersteunt Complex Objecten in de vorm van klassen[2].	Ondersteunt geen Complex Objecten. Echter Oracle ondersteunt complex objecten in de vorm van Oracle objecten[3].
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf - Hoofdstuk 1 paragraaf 3 [2] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 9, *pagina 268* [3] http://www.hk8.org/old_web/oracle/prog2/ch18_02.htm	

Concept:	Object identity	
Beschrijving:	Elk object binnen een OODMS is een object dat onafhankelijk is van zijn waarde. De identity van een object wordt belangrijk wanneer twee of meerdere objecten equivalent zijn aan elkaar. Er zijn twee manieren dat objecten equivalent kunnen zijn. De twee objecten zijn identiek aan elkaar. De andere manier is dat de objecten dezelfde waarden hebben. Er zijn situaties denkbaar wanneer het handig is als een object geïdentificeerd kan worden. Bijvoorbeeld wanneer er de vraag komt of een object wordt gedeeld door verschillende objecten. Binnen een OODBMS kan worden gekeken naar de pointer verwijzing. Het zou ook kunnen door een object een extra identificatie te geven. Een vergelijkbare oplossing wordt gebruikt in een relationele database door gebruik te maken van een Primaire sleutel.	
OODBMS	**OOPL**	**RDBS**
O2 geeft elk nieuw object een unieke identity die zijn hele leven van hem blijft[1].	Er is geen andere object identity, dan de verwijzing van een pointer[2].	Voor object identity heeft een relationele database, Primary key[3].
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf - Hoofdstuk 1 [2] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 8, *pagina 244* [3] [Codd]	

Concept:	Encapsulation/*Information Hiding*	
Beschrijving:	Volgens [M. Atkinson et al. 1995] zijn er twee redenen voor encapsulation. De eerste is omdat er een scheiding moest komen tussen de specificatie en implementatie van een operatie. Encapsulation is ontstaan door de abstracte data types. Een abstracte data type heeft een interface en een implementatie deel. De interface is de specificatie en de implementatie deel is de uitwerking van de implementatie. Dit wordt nu ook Information Hiding genoemd. De andere rede is dat een systeem modulair moet zijn. Een systeem wordt opgebouwd uit verschillende onderdelen. Zo kan een systeem opgebouwd worden door verschillende modulen te koppelen.	
OODBMS	**OOPL**	**RDBS**
O2 kan omgaan met klassen	Een OOPL ondersteunt	RDB ondersteunt geen

geschreven in onder andere Java en C++. Hierdoor ondersteunt O2 encapsulation[1].	encapsulation in de vorm van data hiding[2].	encapsulation. Er is geen onderscheidt tussen definitie en implementatie van de view.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf - Hoofdstuk 2 paragraaf 2 [2] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 9, *pagina 269*	

Concept:	Type	
Beschrijving:	In [M. Atkinson et al. 1995] wordt er een onderscheid gemaakt tussen OOPL die gebruik maken van twee soorten van OO systemen: Types en Classes. Echter is er geen keuze in de programmeertaal, want Mattic maakt gebruik van Delphi en hier wordt ook niet van afgeweken voor deze opdracht. Het OO systeem van Delphi komt overeen met de beschrijving van het Type in [M. Atkinson et al. 1995]. Daarom zal alleen het Type systeem worden beschreven. Het type systeem is gebaseerd op overeenkomsten tussen objecten binnen een verzameling. De notatie komt overeen met die van een abstract Type. Een object bestaat uit een interface en een implementatie kant. De implementatie kant bestaat uit een data deel en een operatie deel. De data kan worden gebruikt voor het beschrijven van de structuur. Het operatiedeel bestaat uit methodes en procedures waarmee de handelingen worden beschreven van het object.	
OODBMS	**OOPL**	**RDBS**
O2 ondersteunt Type, want de klassen van C++ kunnen worden gebruikt door O2[1].	Een OOPL, zoals C++ ondersteunt Type, dit wordt ook aangegeven [M. Atkinson et al. 1995].	RDB ondersteunt geen Type. Een tabel kan wel worden gezien als een klasse.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf - Hoofdstuk 2	

Concept:	Type hiërarchieën	
Beschrijving:	Overerving is een krachtig mechanisme van een OOPL. De broncode van een klasse kan hierdoor opnieuw worden gebruikt. Volgens Atkinson [M. Atkinson et al. 1995] zijn er vier verschillende soorten van overerving te onderscheiden: substition inheritance, inclusion inheritance, constraint inheritance en specialization inheritance. Substition inheritance: De kind-klasse van A, A' kan meer operaties uitvoeren dan A. Inclusion inheritance: Een object van de kind-klasse A' kan ook een object zijn van de ouder-klasse A. Constraint inheritance: De kind-klasse van A, A', heeft meer beperkingen dan de ouder-klasse A. Specialization inheritance: De kind-klasse van A, A' heeft specifieke data en/of operaties die niet voorkomen bij de ouder-klasse A.	
OODBMS	**OOPL**	**RDBS**
O2 ondersteunt types hiërarchieën zolang deze ook te maken zijn in een OOPL[1].	Een OOPL zoals C++ ondersteunt types hiërarchieën door middel van overerving[2].	RDB ondersteunt geen Type hiërarchieën. Echter kunnen verschillende hiërarchieën van tabellen gemaakt worden. Subtype hiërarchieën zijn te maken, maar het gedragsaspect ontbreekt waardoor er geen sprake is van overerving van

		gedrag.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf - Hoofdstuk 2 [2] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 11, *pagina 337*	

Concept:	Overriding, overloading en late binding	
Beschrijving:	Overriding en overloading hebben te maken met hergebruik van een naam van een procedure/methode/functie geïmplementeerd in de ouder-klasse door een kind-klasse. Overriding en overloading lijken veel op elkaar, maar er zijn verschillen[1]: Bij overloading hebben de functies dezelfde naam, maar ze moeten verschillende typen of aantallen argumenten hebben. Bij overriding hebben de functies in de basis-klasse en in de afgeleide klasse dezelfde naam en precies dezelfde argumenten. Om gebruik te maken van deze functionaliteit worden functienamen niet gebonden tijdens het compileren, maar worden achterhaald tijdens run time. Dit wordt late binding of dynamische binding genoemd.	
OODBMS	**OOPL**	**RDBS**
Ja, O2 ondersteunt Overriding, overloading en late binding zolang deze ook ondersteund zijn in een OOPL[2].	Een OOPL, zoals C++ , Java en Delphi, ondersteunt Overriding, overloading en late binding[1].	Wordt niet ondersteund door een relationele database.
Note:	[1] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 11, *pagina 342, pagina* [2] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf - Hoofdstuk 2	

Concept:	Ad Hoc Query Facility	
Beschrijving:	De database kan/moet een Ad Hoc Query Facility ondersteunen. Wanneer deze ondersteund is, kan via de taal van de Ad Hoc Query Facility informatie worden opgevraagd uit de database. Volgens [M. Atkinson et al. 1995] moet voldoen aan de volgende eisen: Het moet van een hoog niveau zijn. Dit betekent dat de gebruiker in enkele woorden kan aangeven wat eruit de database wordt opgevraagd. Het moet effectief zijn. De query moet zichzelf kunnen optimaliseren. Het moet applicatie onafhankelijk zijn, dus moet het werken op alle databases en met alle Types die door de gebruiker gedefinieerd wordt.	
OODBMS	**OOPL**	**RDBS**
Ja, O2 ondersteunt een Ad Hoc Query Facility in de vorm van OQL[1].	Wordt niet ondersteund door OOPL.	Wordt ondersteund door RDB in de vorm van SQL.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/oql.pdf	

Concept:	Computational completeness	
Beschrijving:	Onder computational completeness wordt verstaan dat alle berekenbare functies kunnen worden uitgedrukt door gebruik te maken van de DML van het database systeem [M. Atkinson et al].	
OODBMS	**OOPL**	**RDBS**
Wanneer OQL op basis van een latere SQL, die wel computational compleet is, is gemaakt. Dat is de OQL ook computational complete [1].	Wordt niet ondersteund door OOPL.	SQL is niet computational complete, echter het relationeel model is dat wel. Latere SQL uitbreidingen zijn wel computational complete. Deze standaard is echter nog niet wijd verspreid.

Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/oql.pdf

Concept:	Extensibility	
Beschrijving:	Een database heeft een verzameling van gedefinieerde types. Deze types kunnen gebruikt worden door de programmeur. Deze verzameling moet ook uitgebreid kunnen worden. De nieuwe Types kunnen op dezelfde manier worden gebruikt als de gedefinieerde Types.	
OODBMS	**OOPL**	**RDBS**
O2 is Extensibility, want datatypes kunnen worden gemaakt door de ontwikkelaar zelf en gebruikt worden[1].	Het uitbreiden van bestaande Types/klassen wordt ondersteund door OOPL[2].	Dit staat niet beschreven in [Codd], maar extensibility wordt wel ondersteund door Oracle[3].
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf - Hoofdstuk 2 [2] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 11, *pagina 337* [3] http://sales.esicom.com/sales/oracle/network.816/a77230/concepts.htm	

Concept:	Schema evolution	
Beschrijving:	Heeft te maken met het aanpassen van het klasse-schema. Een OODBMS moet alle klassen aanpassen die worden beïnvloed wanneer er bepaalde klasse wordt verwijderd uit het klasse-schema. Operaties die te maken hebben met schema evolution zijn: het creëren, herbenoemen en verwijderen van een klasse. Deze operaties moeten het database schema consistent houden.	
OODBMS	**OOPL**	**RDBS**
Binnen O2 is het mogelijk om het klasse schema aan te passen[1].	Het doorvoeren van veranderingen in het schema moet door de ontwikkelaar zelf worden afgehandeld.	Er is dan geen klasse schema om aan te passen in een RDB, maar het is wel mogelijk om de schema van de tabellen zelf aan te passen. Dat kan worden gezien als schema evolution.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf - hoofdstuk 8	

Concept:	Persistence	
Beschrijving:	De gegevens van een programma is persistent, wanneer deze gegevens opnieuw kunnen gebruikt worden in een nieuw proces. Alle objecten binnen een OODBMS moeten persistent zijn, zonder dat deze expliciet worden verplaatst of gekopieerd [M. Atkinson et al].	
OODBMS	**OOPL**	**RDBS**
Om gegevens van de klassen persistent te maken moeten de zelfgemaakte klasse, een kind-klassen worden van een persistent klasse van O2[1].	Neen, klassen zijn niet persistent. Echter, kunnen de klassen van de programmeertaal Eiffel wel persistent zijn[2].	Alle gegevens in een RDB zijn persistent.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/cppbdgui.pdf [2] http://www.maths.tcd.ie/~odunlain/eiffel/eiffel_course/eforb.htm	

Concept:	Secondary storage management
Beschrijving:	Dit is een onderdeel van een "klassiek" database systeem. Het bestaat

	uit verschillende mechanismen, zoals index management, data clustering, data buffering, acces path selection en query optimization. Deze secondary storage management is niet zichtbaar voor de gebruiker. Het doel van deze secondary storage management systeem is gericht op een betere performance. De gebruiker hoeft zelf geen onderhoud te pleegt op gegevens in de main memory en gegevens op disk.

OODBMS	**OOPL**	**RDBS**
Dit wordt ondersteund in O2 doordat er verschillende RDB kunnen worden aangesproken om zo de gegevens op die databases te plaatsen[1].	Nee, wordt niet ondersteund.	Dit is niet nodig.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/dbacscpp.pdf	

Concept:	Concurrency
Beschrijving:	Concurrency heeft alles te maken met data integrity. Het moet voorkomen worden dat er conflicten ontstaan terwijl meerdere gebruikers dezelfde gegevens aanpassen. Concurrency omvat meerdere concepten die hieronder zullen worden besproken. • *Transactions* Handelingen uitvoeren op gegevens in een database. Een database die meerdere gebruikers tot zijn gegevens toe laat moet dus ook meerdere transacties kunnen afhandelen. • *Locking* Locking wordt gebruikt wanneer een gebruiker data gaat aanpassen in een database systeem. De data waarmee die gebruiker werkt, wordt tot de gebruiker de data vrijgeeft, als bezet aangemerkt. • *Distribution* Distribution heeft te maken met het verspreiden van gegevens die in de database staan. Distribution kan tussen verschillende databases plaatsvinden of tussen gebruikers en een database. o *Replication* Replication heeft te maken met het synchroniseren van gegevens tussen twee databases. Het repliceren van twee databases kan op dagelijkse of wekelijkse basis plaats vinden. o *Multi User* De huidige databases zouden met meerdere gebruikers moeten kunnen omgaan. Het gaat dan om een centrale database waarmee alle users werken. o *Synchronisation* Live synchronisatie tussen de client en de server. Het verschil met replicatie is dat hier elke verandering binnen de database van de client meteen wordt verwerkt op de database van de server.

OODBMS	**OOPL**	**RDBS**
O2 heeft de mogelijkheid om meerdere clients toegang te geven tot de gegevens, om gebruik te maken van het transaction systeem moet de programmeur een daarvoor bestemde klasse gebruiken.	Een OOPL ondersteunt dit niet.	Een RDBMS ondersteunt alles wat genoemd wordt onder concurrency.

Distribution wordt ook ondersteund door O2. Replication wordt ook door O2 ondersteund, het is dus mogelijk om twee O2 databases gelijk aan elkaar te maken. Multi user wordt ook ondersteund door O2. Synchronisation wordt gedaan via de notification en transaction systeem[1].		
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2notif.pdf	

Concept:	Recovery	
Beschrijving:	Hier gaat het om het herstellen van gegevens na een crash. De computer waarop de database staat moet dan opnieuw worden gestart, zonder dat er gegevens verloren gaan.	
OODBMS	**OOPL**	**RDBS**
O2 ondersteunt recovery op een de volgende manieren: Bij het uitvallen van de stroom, dan kan het O2-systeem weer opgestart worden zonder probleem. Wanneer er meerdere oorzaken of gevolgen zijn van de crash, zoals een fout op de disk, dan moet er gebruik worden gemaakt van de laatste backup van het O2-systeem[1].	Recovery van gegevens wordt niet meteen ondersteund door een standaard programmeertaal; dus de ontwikkelaar moet hiervoor zelf iets bedenken tijdens de ontwikkeling van het systeem.	De meeste relationele databasesystemen hebben voorzorgsmaatregelen voor de recovery van gegevens.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/admingui.pdf	

Concept:	Version	
Beschrijving:	Een object georiënteerde database kan ook versiebeheer ondersteunen. Het is dus mogelijk om meerdere versies van een Type te ontwikkelen binnen een OBDMS.	
OODBMS	**OOPL**	**RDBS**
O2 komt met een speciale library die verschillende versies van objecten ondersteunt[1].	Bij applicaties waarin ontworpen wordt is het gebruikelijk om versiebeheer toe te passen. Echter zijn hier verschillende applicaties voor.	RDBMS ondersteunt geen verschillende versie van gegevens.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2verson.pdf	

Concept:	Relation	
Beschrijving:	R is het Cartesisch product van S1 x S2 x S3 … X Sn. S is een Set van n-tuples.	
OODBMS	**OOPL**	**RDBS**
Sets en tuples zijn een onderdeel van de OODBMS, zie ook complex objecten[1].	Sets en tuples kunnen worden gemaakt door gebruik te maken van een bepaalde library die deze mogelijkheden aanbiedt. Of er daarna een relatie te	Ja.

	leggen is, zoals R beschrijft is afhankelijk van de gebruikte library.	
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf	

Concept:	Relationships	
Beschrijving:	Is het tegenovergestelde van relation met als eigenschap dat de domeinen niet zijn geordend.	
OODBMS	**OOPL**	**RDBS**
In O2 is dit misschien te vergelijken met een set[1].	Hier wordt niet gesproken over domeinen.	Ja, een RDB kent relationships.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf	

Concept:	Set	
Beschrijving:	Een verzameling van tuples.	
OODBMS	**OOPL**	**RDBS**
O2 ondersteunt set[1].	Een set kan worden gebruikt in een OOPL door gebruik te maken van een library waarin een set gedefinieerd is.	Hierin wordt de set ondersteund.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf	

Concept:	Tuple	
Beschrijving:	TUPLE1 bestaat uit het eerste element van S1, S2, …, Sn. TUPLE2 bestaat uit het tweede element van S1, S2, …, Sn. TUPLEn bestaat uit het n-de element van S1, S2, …, Sn.	
OODBMS	**OOPL**	**RDBS**
O2 ondersteunt een tuple[1].	Sets en tuples kunnen worden gemaakt door gebruik te maken van een bepaalde lib die deze mogelijkheden aanbiedt. Een tuple zoals hierboven staat beschreven is vergelijkbaar met een object van een klasse.	Ja.
Note:	[1] http://www.csd.uwo.ca/courses/CS411a/pdfO2manuals/o2cref.pdf	

Concept:	Row	
Beschrijving:	Een row is de weergave van n-tuple van R	
OODBMS	**OOPL**	**RDBS**
Een row is te vergelijken met een object van een klasse.	Een row is te vergelijken met een object van een klasse.	Ja, wordt ondersteund.
Note:	[1] [verwijzing handleiding O2] [2] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 11, *pagina 337*	

Concept:	Column	
Beschrijving:	De columns komen overeen met de ordening van S1, S2, …, Sn van de domeinen die in R zijn gedefinieerd.	
OODBMS	**OOPL**	**RDBS**
De column is te vergelijken met constanten en variabelen van een klasse.	De column is te vergelijken met constanten en variabelen van een klasse.	Ja, wordt ondersteund.

Note:	[1] [verwijzing handleiding O2] [2] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 11, *pagina 337*

Concept:	Domein	
Beschrijving:	Elke element van S hoort bij een domein en dit domein moet een duidelijke naam hebben.	
OODBMS	**OOPL**	**RDBS**
Het domein is te vergelijken met een klasse.	Het domein is te vergelijken met een klasse[1].	Ja, wordt ondersteund.
Note:	[1] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 11, *pagina 337*	

Concept:	Component	
Beschrijving:	De afbeelding van de relation wordt component genoemd. De component bestaat uit domeinen.	
OODBMS	**OOPL**	**RDBS**
Dit is te vergelijken met een afbeelding van de constanten en variabelen van een klasse.	Dit is te vergelijken met een afbeelding van de constanten en variabelen van een klasse.	Ja, wordt ondersteund.
Note:		

Concept:	Role name	
Beschrijving:	Wanneer de component uit twee of meerdere domeinen bestaat, dan kan via de role naam worden aangegeven wat de rol binnen de component is van het domein.	
OODBMS	**OOPL**	**RDBS**
Dit is te vergelijken met een pointer verwijzing naar een object.	Dit is te vergelijken met een pointer verwijzing naar een object.	Dit wordt ondersteund zoals beschreven staat.
Note:	[1] [verwijzing handleiding O2]	

Concept:	Active domein	
Beschrijving:	Is de set van waarden uit de database, dat getoond wordt op een bepaalde tijd.	
OODBMS	**OOPL**	**RDBS**
Dit is te vergelijken met de gegevens van een object tonen.	Dit is te vergelijken met de gegevens van een object tonen.	Dit wordt ondersteund zoals beschreven staat in [Codd].
Note:		
Concept:	Primary key	
Beschrijving:	Eén domein of een combinatie van domeinen van een relation hebben waarden die een unieke identificatie zijn voor die tuples.	
OODBMS	**OOPL**	**RDBS**
Zie object identity.	Dit bestaat niet in het OO schema.	Wordt ondersteund.
Note:		

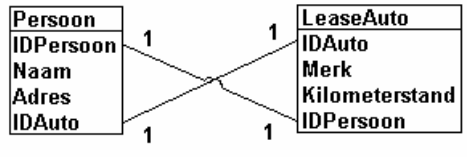
Concept:	Foreign key	
Beschrijving:	Eén domein of een combinatie van domeinen van een relation waarvan het domein een deelverzameling is van de primaire key van de relatie.	
OODBMS	**OOPL**	**RDBS**
Pointer verwijzing naar een klasse.	Pointer verwijzing naar een klasse.	Wordt ondersteund.
Note:		

Concept:	Nonsimple Domein		
Beschrijving:	Dat is een domein waarvan een element met als waarde een relation.		
OODBMS		**OOPL**	**RDBS**
Dit kan worden vergeleken met een pointer verwijzing naar een klasse van een andere klasse.		Dit kan worden vergeleken met een pointer verwijzing naar een klasse van een andere klasse.	Wordt ondersteund.
Note:	[1] [verwijzing handleiding O2] [2] Aan de slag met C++, *Gertjan Laan*, Hoofdstuk 11, *pagina 337*		

Appendix 3. – Impedance Mismatch

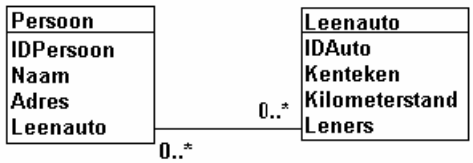
Veel van de ontwikkelaars die een relationele database gebruiken om gegevens in op te slaan, kunnen te maken krijgen met het probleem dat impedance mismatch genoemd wordt. Dit probleem ontstaat doordat de modulering vanuit het Object georiënteerd,OO,-oogpunt wordt gedaan [W Keller]. In het OO-model zijn concepten mogelijk die niet mogelijk zijn in het relationeel model [Idris]. Veel bedrijven gebruiken UML als modeleertaal. Tijdens het ontwerpen wordt er geen rekening gehouden met de gebruikte relationele database. Hierdoor ontstaan moeilijkheden met de mapping van gebruikte klassen op de tabellen van de database. Het niet kunnen mappen van de klassen op tabellen wordt ook wel de impedance mismatch genoemd. De fouten treden op wanneer relaties tussen de klassen worden vastgelegd. Bij het voorbeeld op figuur 1 gaat de mapping wel goed:

Een klasse persoon heeft een verwijzing naar één klasse lease auto.



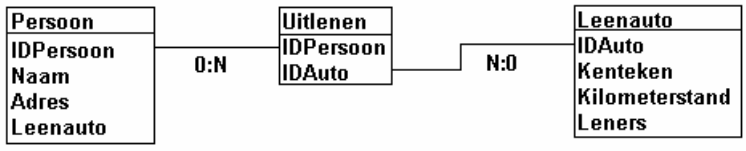
Figuur 1 Klasse diagram lease auto

De mapping van het volgende voorbeeld gaat niet goed:
In een klassendiagram zou dit worden aangegeven als een klasse persoon met daarin een lijst met leenauto's en omgekeerd, zoals op figuur 2.



Figuur 2 klasse diagram van lener en leen auto

De mapping van deze klassen is niet mogelijk op een relationeel model. In een relationeel model zullen 3 tabellen zijn. Er is een tabel voor Persoon, een tabel om een Persoon te koppelen aan verschillende Leenauto's en een tabel waarin de verschillende Leenauto's staan, zie ook figuur 3.



Figuur 3 relationele schema van lener en leen auto

Er zijn meer voorbeelden om aan te geven dat de mapping van UML klasse schema op een relationeel schema niet goed gaat. Dit komt doordat het UML klasse schema verschillende regels heeft een relationele schema heeft (zie ook de scriptie van Idris).

3.01 Voorkomen en omgaan met de Impedance Mismatch

De manier waarop Mattic Software ontwikkelt, zie hoofdstuk 5 paragraaf 1.3, voorkomt dat de Impedance Mismatch ontstaat. Mattic moduleert software niet aan de hand van UML, maar vanuit het relationeel schema. Hierdoor worden de klassen gemaakt met als basis de tabellen die bestaan in het relationele schema. Dus als het Relationele schema gelijk is aan die op figuur 3, dan worden er ook drie klassen gemaakt. Hierdoor wordt

voorkomen dat de mapping van de klassen met de tabellen van de database, de impedance mismatch wordt zo voorkomen.
J2EE genereert ook klassen op basis van het relationele model. Echter wordt niet zoals bij Mattic de hele database geladen, maar het deel dat gebruikt wordt [Yassin].
In [W Keller] staat als een andere manier om de impedance mismatch te, het gebruik maken van een object georiënteerde database, OODB. De mapping van de klassen in de oodb moeten wel gelijk zijn aan de klassen gebruikt door het programma dat gebruikt maakt van de OODB.

Appendix 4. Literatuurlijst

[M. Atkinson et al] *Atkinson M. et al,* The Object-Oriented Database System Manifesto*, Proc. First International Conference on Deductive and Object-oriented Databases,* Kyoto*, Japan,* 1989

In dit document wordt geprobeerd om een omschrijven te geven waaraan een object georiënteerde database moet voldoen. Er zijn een aantal minimale eisen die worden gesteld aan een systeem voordat het een OO database systeem is. Daarnaast zijn nog elke opties die eventueel kunnen worden toegevoegd. Dan zijn er nog elke opties die helemaal openstaan. Deze opties hebben meer te maken met het ontwerp en dergelijke.

[Elisa Bertina et al] *Elisa Bertina et al.,* Object-Oriented Database Management Systems: Concepts and Issues, *Volume 24*, Issue 4 (April 1991) - Special issue on instruction sequencing, *Pages: 33-47*, IEEE Computer Society Press Los Alamitos, CA, USA*,* 1991

Dit document licht enkele concepten toe bij issues die spelen bij het ontwikkelen van een oodbms. Er worden verschillende oodbms bekijken en verteld wat de verschillen zijn tussen de oodbms.

[W Keller] *Wolfgang Keller*, Persistence Options for Object-Oriented Programs*, http://www.objectarchitects.de/ObjectArchitects/events/OOP2004/PersistenceOptionsOOP2004e.pdf*

In dit document staat een voorstel om objecten te mappen op een relationele database. Het document is een inleidend verhaal. Het document gaat niet diep in op de gebruikte technieken voor het mappen. Er staat een voorstel in voor mapping voor de programmeertaal Java. Daarnaast wordt ook gekeken hoe de ontwikkeling van mapping is bij de programmeertaal .NET.

[Codd] *E. F. Codd*, A Relational Model of Data for Large Shared Data Banks*, Reprinted from Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387. Copyright © 1970, Association for Computing Machinery, Inc.*

Dit document is de basis voor elk relationele database. Codd legt in dit document het relationeel model uit.

[0. DEUX et al.].0. DEUX et al., The Story of O2, *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 2. NO. I, MARCH 1990*

In dit document wordt een beschrijving gegeven van de object georiënteerde database O2.

[Idris].
Scriptie van Idris.

[Yassin].
Scriptie van Yassin.

[Laan] *Gertjan Laan,* Aan de slag met C++, eerste druk: januari 1997. *Uitgeverij Pim Oets*
Dit boek is een introductie tot C++.