

**Eine Umsetzung von  
OQL-Anfragen für Legacy Datenbanken**

*Autor: Oriel Maute*

*Stand: 20. Juni 2001*

## 1. Einführung

Software wird heutzutage überwiegend nach objektorientierten Vorgehensweisen entwickelt und mit Hilfe von objektorientierten Programmiersprachen implementiert. Konsequenterweise sind objektorientierte Datenbanksysteme für die Speicherung von persistenten Objekten zu verwenden. Allerdings gibt es zahlreiche Gründe, die auch heute noch für den Einsatz relationaler Datenbanksysteme sprechen (siehe [Wan96]).

Daher steht der objektorientierten Programmiersprache Java einerseits häufig der Einsatz eines relationalen Datenbanksystems andererseits gegenüber.

Für die Koppelung dieser Konzepte, dem objektorientierten Datenmodell der Programmiersprache und dem relationalen Datenmodell des Datenbanksystems wird die objektorientierte Programmiersprache um ein objektorientiertes Datenbankmodell (*Persistenzframework*) erweitert. Dieses verbirgt sowohl das relationale Datenbankmodell wie auch die zugehörige Datenbanksprache, meist SQL, vor dem Entwickler. Er bewegt sich somit ausschließlich im objektorientierten Datenmodell seiner Programmiersprache.

Ein solches objektorientiertes Datenbankmodell (*Persistenzframework*) hat unter anderem die Aufgabe, den Entwickler bei der Suche nach persistenten Objekten und Daten zu unterstützen. Hierfür gibt es unterschiedliche Anfragesprachen, die je nach Einsatzzweck folgende Vor- und Nachteile (siehe [Wan97]) aufweisen:

- Rein navigierende Anfragesprachen sind zwar problemlos in SQL-Anweisungen transformierbar, besitzen allerdings durch mangelnde Mächtigkeit oft zu wenig Ausdrucksmöglichkeiten. Anfragen müssen daher unter Zuhilfenahme der Navigationspfade programmiert werden, was wegen der Flut von erzeugten SQL-Anweisungen zu einer völlig inakzeptablen Performance führt.
- Bei den assoziativen Objektanfragen dienen Musterobjekte als Träger der Suchbedingung. Ausgehend von einem solchen Musterobjekt wird automatisch eine Anfrage formuliert, die alle Objekte ermittelt, die „so ähnlich“ wie das Musterobjekt sind. Neben den beschreibenden Attributen eines Objekts werden auch referenzierende Attribute zu Komponentenobjekten ausgewertet. Ein Vorteil von assoziativen Objektanfragen besteht darin, dass sich Anfragen ausschließlich im Datenmodell der objektorientierten Programmiersprache bewegen. Als Nachteil ist die mangelnde Mächtigkeit zu nennen. So ist es beispielsweise sehr schwierig, in einem Schema, in dem Personen und Adressen durch eine Beziehung miteinander verknüpft sind, die Personen zu finden, die keine Adresse besitzen.
- Die objektorientierte Anfragesprache OQL (*Object Query Language*) der ODMG (*Object Data Management Group*) fügt sich, auf Grund der zugehörigen Language Bindings, anstandslos in das Datenmodell der Programmiersprache ein (siehe [Cat98]). OQL ist eine sehr mächtige und standardisierte Anfragesprache die jedoch nur schwierig für Persistenzframeworks umgesetzt werden kann.

## **2. Zielsetzung**

Dieser Beitrag beschreibt, wie die objektorientierte Datenbankansprache OQL des ODMG 2.0 Standards bei relationaler Datenhaltung (Legacy-Datenbanken) verwendbar gemacht werden kann. Es wird angegeben, inwieweit OQL-Anfragen, für eine effiziente Auswertung, direkt in semantisch äquivalente SQL-Anfragen übersetzt werden können. Semantisch äquivalent bedeutet im folgenden, dass beide Anfragen dieselben Daten zurückliefern, jedoch bei der OQL-Anfrage auf Objektebene (d.h. in Form von Objekten und Kollektionen) und bei der SQL-Anfrage auf Relationenebene (d.h. als Mengen von Tupeln). Zudem wird gefordert, dass das Ergebnis der SQL-Anfrage, evtl. unter Verwendung von Typinformationen, eindeutig in das Ergebnis der OQL-Anfrage umgewandelt werden kann.

Für die Übersetzung von OQL nach SQL stehen zur Laufzeit zugreifbare Informationen über das Objektmodell (Metamodell) und dessen relationaler Abbildung (OO-ER-Mapping) zur Verfügung. Zu diesen Informationen gehören beispielsweise Angaben über die Vererbungshierarchien, über die Beziehungen zwischen Klassen und über die physische Abbildung von Objekten in relationalen Strukturen.

## **3. Lösungsansatz**

Ein Lösungsansatz für die Umsetzung von OQL-Anfragen besteht darin, die OQL-Anfrage mit Hilfe eines Compilers, in eine semantisch äquivalente SQL-Anfrage zu übersetzen. Hinsichtlich des Laufzeitverhaltens ist dieses Vorgehen geeignet, da nur eine einzige SQL-Anfrage erzeugt wird, welche, dank den Optimierungsmechanismen des relationalen Datenbanksystems, effizient ausgewertet werden kann. Leider lassen sich nicht alle OQL-Anfragen in semantisch äquivalente SQL-Anfragen übersetzen. Dies liegt u.a. daran, dass die Sprache OQL das Aufrufen von Methoden fachlicher Klassen in einer Anfrage erlaubt, während dies in einer relationalen Anfragesprache wie SQL überhaupt nicht möglich ist.

Des weiteren unterstützt OQL das erweiterte NF<sup>2</sup>-Modell, d.h. Anfragen können strukturierte und geschachtelte Ergebnisse zurückliefern. Dies ist im einfachen relationalen Modell nicht vorgesehen und auch nicht erwünscht, da Relationen per Definition in erster Normalform sein müssen. Folglich können OQL-Anfragen, die fachliche Methodenaufrufe verwenden oder auch geschachtelte Ergebnisse liefern, im allgemeinen weder in eine einzige noch in mehrere hintereinander auszuführende SQL-Anfragen übersetzt werden.

Trotzdem ist es möglich, auch solche OQL-Anfragen auszuwerten: Man ermittelt dabei die notwendigen Rohdaten, gemeint sind hierbei die Tupel der Relationen, unter Verwendung von sehr einfachen SQL-Anfragen. Anhand dieser Daten kann die OQL-Anfrage mittels Interpretation auf Objektebene ausgewertet werden. Dabei werden allerdings die Optimierungsmechanismen des relationalen Datenbanksystems umgangen; die Folge ist eine relativ ineffiziente Auswertung der Anfrage.

Zusammenfassend weisen die beiden Vorgehensweisen der direkten Umsetzung nach SQL und der Interpretation zwar Vorteile aber auch extreme Nachteile auf. Es liegt nahe, die Vorteile der Verfahren auszuschöpfen, in dem die Verfahren miteinander kombiniert werden. Dadurch ergibt sich eine neue Vorgehensweise, die wie folgt beschrieben werden kann:

Die OQL-Anfrage wird, sofern möglich, in eine einzige semantisch äquivalente SQL-Anfrage übersetzt. Ist eine Übersetzung nicht möglich, so wird die OQL-Anfrage unter zu Hilfenahme von einfachen SQL-Anfragen interpretiert.

Die kombinierte Vorgehensweise hat den Vorteil, dass der Software-Entwickler bzw. Anwender nicht wissen muss, welche seiner Anfragen direkt nach SQL übersetzt werden können und welche nicht, denn es wird stets der volle Sprachumfang von OQL unterstützt. Allerdings erhöht sich die Ausführungszeit in den Fällen, in denen eine Interpretation notwendig ist. Dies ist aber der unumgängliche Preis für eine volle OQL-Unterstützung.

Eine mögliche Umsetzung von OQL-Anfragen bei relationaler Datenhaltung erfolgt in den in Abbildung 1 aufgezeigten drei Phasen:

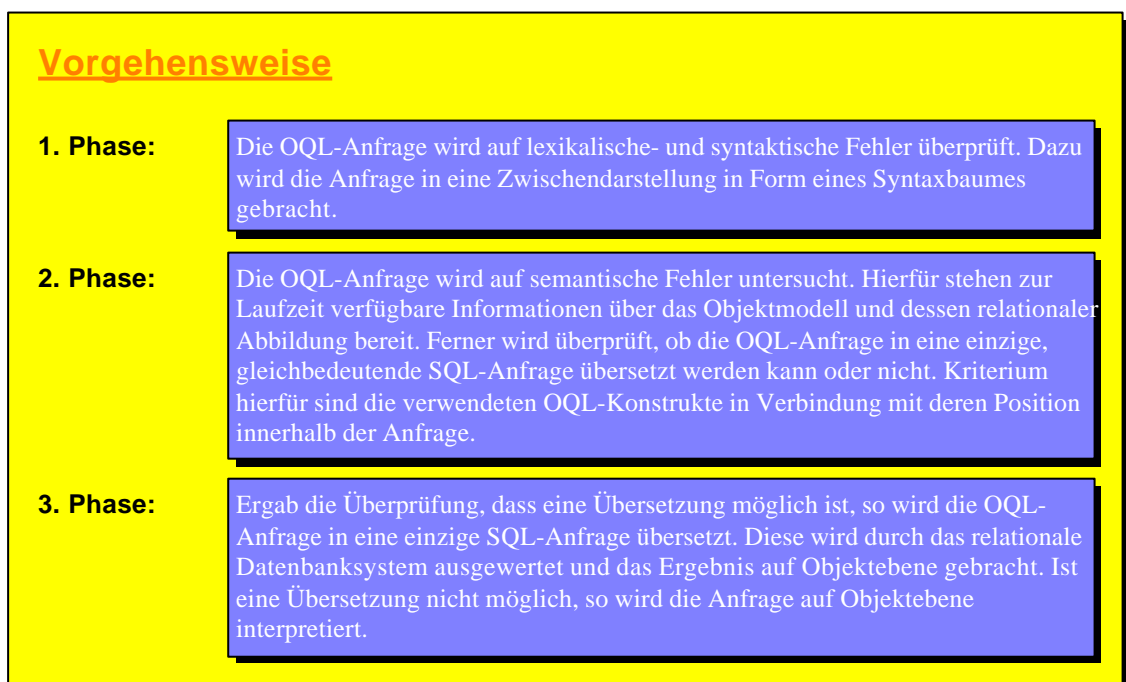


Abbildung 1: Mögliche Vorgehensweise zur Umsetzung von OQL-Anfragen

Die Phasen geben an, wie oft der Syntaxbaum traversiert werden muss. Die Phasen eins und zwei können, wie im Compilerbau üblich, auch zu einer einzigen Phase zusammengefaßt werden.

Im folgenden wird dargestellt, wie ausgewählte OQL-Konstrukte in SQL dargestellt werden können.

## 4. Direkte Transformation von OQL nach SQL

In diesem Kapitel wird anhand von Beispielen gezeigt, wie spezielle OQL Sprachkonstrukte in semantisch äquivalente SQL Konstrukte übersetzt werden können. Die Beispiele beruhen auf dem in Abbildung 2 angegebenen Objektmodell und dessen relationaler Abbildung. Das Objektmodell besteht aus den drei Klassen Trainer, Mannschaft und Spieler. Zwischen den Klassen Trainer und Mannschaft gibt es eine 1:1-Beziehung. Sie gibt an, welcher Trainer welche Mannschaft trainiert. Ferner existiert eine 1:N-Beziehung zwischen den Klassen Mannschaft und Spieler. Sie beschreibt die Spieler einer Mannschaft.

Für die persistente Speicherung von Objekten wird jeder Klasse eine eigene Tabelle (Relation) im relationalen Datenbanksystem zugeordnet. Entsprechend wird jedem beschreibenden Attribut eine Tabellenspalte in der korrespondierenden Tabelle zugeordnet. Primärschlüssel (OID, *Object Identification*) werden automatisch von dem Persistenzframework erzeugt und in einer dafür vorgesehenen Spalte gespeichert. Referenzierende Attribute werden mit Hilfe von Fremdschlüsseln abgebildet. Bei 1:1-Beziehungen kann der Fremdschlüssel sowohl in der Tabelle der Quell- als auch in der Tabelle der Zielklasse enthalten sein; d.h. die Spalte `TRAINERID` der Relation `MANNSCHAFT` könnte auch in der Relation `TRAINER` abgelegt werden.

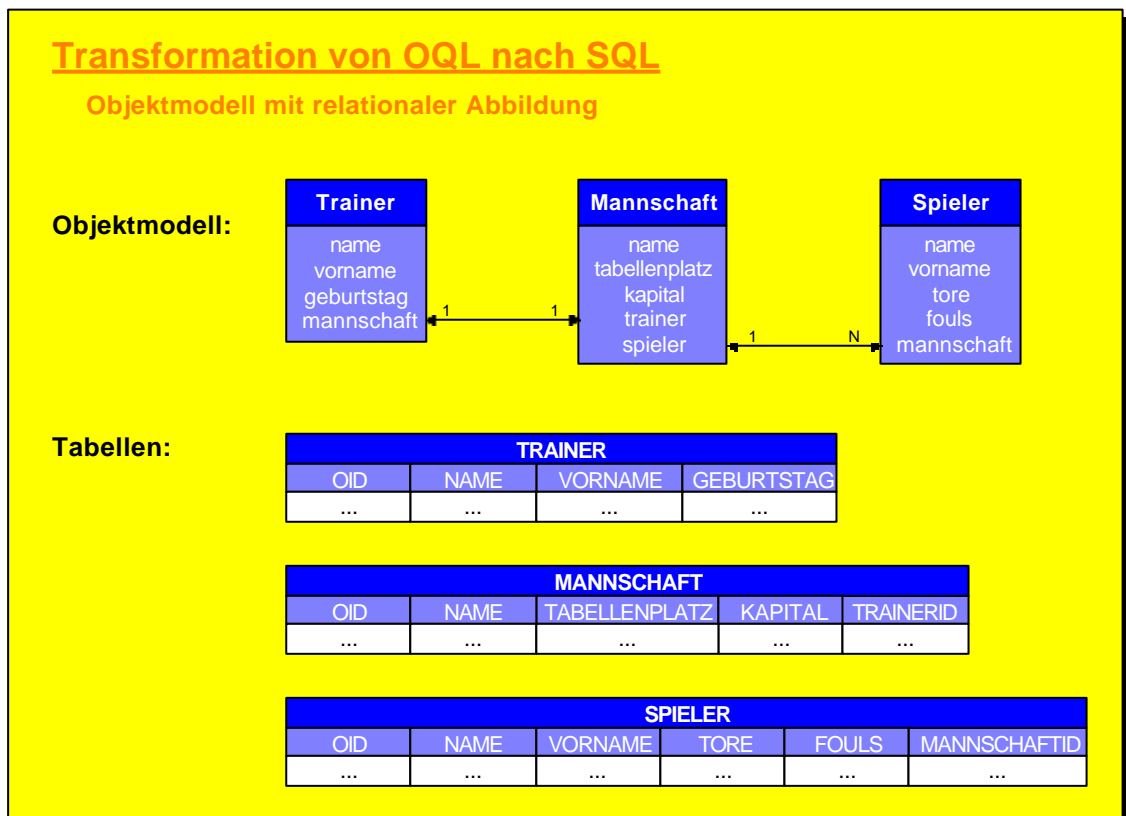


Abbildung 2: Beispiel Objektmodell mit relationaler Abbildung

Der Zugriff auf die Objekte einer objektorientierten Datenbank erfolgt in der Sprache OQL über sogenannte Wurzelobjekte (*root objects*), von denen die restlichen Objekte der

Datenbank erreicht werden. Den Wurzelobjekten werden Namen zugeordnet, über die sie angesprochen werden. Im folgenden sollen per Definition alle persistenten Klassen als Wurzelobjekte dienen. Der Klassenname soll dabei die Menge aller Objekte der jeweiligen Klasse bezeichnen.

Viele einfache OQL-Anfragen können nahezu direkt nach SQL übersetzt werden. Dazu muss lediglich der Klassenname durch den Relationen- oder Sichtenname und der Attributname durch den Spaltennamen ersetzt werden. Sichtennamen (Views) sind bei der Verwendung von Vererbung notwendig, insbesondere wenn die Daten in den Subklassen oder der Superklasse gespeichert werden und nicht für jede Klasse eine eigene Relation zur Verfügung steht. In solch einem Fall repräsentiert die Sicht die Menge der Objekte der Klasse einschließlich den Objekten der Superklassen.

Ein besonders wichtiges Konzept der Sprache OQL ist die Objektnavigation. Sie ermöglicht es, ausgehend von einem Objekt entlang einer zu-1-Beziehung auf ein anderes Objekt zuzugreifen, insbesondere auch auf die Attribute dieses Objektes. Von diesem Objekt kann wiederum per Navigation entlang einer zu-1-Beziehung auf ein anderes Objekt zugegriffen werden und so fort. Möglich ist aber auch die Navigation ausgehend von einem Objekt entlang einer zu-N-Beziehung zu potentiell mehreren Objekten. In diesem Fall ist das Ergebnis einer Navigation eine Menge von Objekten, die auch leer sein kann. Auf diese Menge kann natürlicherweise keine Objektnavigation mehr ausgeführt werden. Navigationen können ausschließlich auf Objekte und nicht auf Mengen bestehend aus Objekten ausgeführt werden. Somit sind mehrstufige Objektnavigationen nur bei zu-1-Beziehungen anwendbar.

Die Objektnavigation kann, auf Grund der Orthogonalität von OQL, in beliebigen Kontexten der Sprache vorkommen, zum Beispiel innerhalb jeder Select-, From- oder Where-Anweisung. Im weiteren wird gezeigt, wie OQL-Anfragen die Objektnavigationen enthalten in gleichbedeutende SQL-Anfragen umgewandelt werden können.

#### **4.1. Navigation in der Select-Anweisung**

Die Abbildung 3 zeigt eine OQL-Anfrage, die für jede Mannschaft den Namen der Mannschaft sowie den zugehörigen Namen des Trainers dieser Mannschaft zurückliefert. Hierbei bezeichnet `Mannschaft` die Menge aller Mannschaften und `m` die aktuell betrachtete Mannschaftsausprägung.

Das Beispiel verwendet eine ganz einfache Form der Objektnavigation und zwar in der Select-Anweisung, also als Projektion: Von der aktuellen Ausprägung des Mannschaftsobjektes `m` wird entlang der 1:1-Beziehung zu dem korrespondierenden Trainerobjekt mittels `m.trainer` navigiert, von welchem dann der Name zurückgeliefert wird.



Abbildung 3: Objektnavigation in der Select-Anweisung

Für diese OQL-Anfrage kann eine semantisch äquivalente SQL-Anfrage erzeugt werden. Die grundlegende Idee hierzu findet sich im Zwischenschritt: Zu jeder Mannschaft  $m$  wird im zweiten Teil der From-Anweisung (siehe Zwischenschritt) die Menge der zu dieser Mannschaft  $m$  gehörenden Trainer ermittelt. Diese Menge hat bei 1:1-Beziehungen und konsistenter Datenbank die Kardinalität 1. Die jeweils aktuelle und bei 1:1-Beziehungen einzige Ausprägung dieser Menge wird in Abbildung 3 als  $nav1$  bezeichnet.

Durch das Hinzunehmen einer weiteren Menge in der From-Anweisung wird als Ergebnis der From-Anweisung das kartesische Produkt der beiden angegebenen Mengen gebildet. Da bei 1:1-Beziehungen, wie oben bereits bemerkt, die zweite Menge die Kardinalität 1 hat, verändert sich die Anzahl der zurückgelieferten Tupel nicht. Dafür sind die einzelnen Tupel aber größer, denn zu den Mannschaftseinträgen enthält jedes Tupel jetzt auch noch die zugehörigen Trainereinträge. Auf die Mannschaftseinträge kann über den Bezeichner  $m$ , auf die Trainereinträge der aktuellen Mannschaft  $m$  kann über den Bezeichner  $nav1$  zugegriffen werden. Statt der Objektnavigation  $m.trainer$  kann also der Bezeichner  $nav1$  verwendet werden. Nachdem Objekte durch Tabellen, Attributnamen durch Spaltennamen ersetzt wurden, ergibt sich die im Zwischenschritt von Abbildung 3 angegebene SQL-Anfrage. Sie beschreibt die gleiche Ergebnismenge auf Relationenebene, welche die OQL-Anfrage auf Objektebene beschreibt.

Leider ist die im Zwischenschritt angegebene SQL-Anfrage von einem bekannten relationalen Datenbanksystemen nicht auswertbar. Grund hierfür ist die in der From-Anweisung eingeschachtelte Select From Where - Anweisung.

Im weiteren wird die vorläufige SQL-Anfrage in eine semantisch gleichbedeutende SQL-Anfrage umgewandelt, die jedoch von allen relationalen Datenbanksystemen, die SQL unterstützen, ausgeführt werden kann. Hilfreich für diese Umwandlung ist folgende Überlegung:

Im Zwischenschritt wird zuerst eine Restriktion auf der Menge der Trainer durchgeführt und dann das kartesische Produkt mit der Menge der Mannschaften gebildet. Laut

Relationenalgebra ist es aber auch möglich, zuerst das kartesische Produkt und dann die Restriktion auszuführen. Entsprechend dieser Überlegung wurde in Abbildung 3 die generierte SQL-Anfrage erstellt. Hierbei wird zuerst das kartesische Produkt aus der Menge der Mannschaften und der Menge der Trainer gebildet und dann erst die Restriktion durchgeführt, dass der Trainer zu der aktuellen Mannschaft gehört.

Die nach dieser Vorgehensweise generierte SQL-Anfrage ist sehr einfach und kann von allen relationalen Datenbanksystemen, die SQL unterstützen, ausgeführt werden.

Wie bereits erwähnt, wird in diesem Beispiel nur eine ganz einfache Form der Objektnavigation verwendet, trotzdem gibt es einen wichtigen Spezialfall, der beachtet werden muss: Wie in Abbildung 3 gezeigt, wurde die From-Anweisung um eine zweite Menge erweitert. Die jeweils aktuelle Ausprägung dieser Menge wurde als `nav1` bezeichnet. Damit die oben beschriebene Vorgehensweise bei der Objektnavigation funktioniert, muss gewährleistet sein, dass der generierte Bezeichner `nav1` innerhalb der gesamten Anfrage eindeutig ist und auch bleibt. Sollte es beispielsweise in der OQL-Anfrage schon einen Bezeichner `nav1` geben, so muss bei der Umsetzung der Objektnavigation ein anderer Bezeichner als `nav1` erzeugt werden, der innerhalb der Anfrage eindeutig ist.

## **4.2. Navigation in der Where-Anweisung**

Abbildung 4 zeigt nicht mehr nur eine einfache, sondern eine zweifach hintereinander ausgeführte Objektnavigation, die sich diesmal in der Where-Anweisung befindet: Von der aktuellen Spielerausprägung `s` wird entlang einer 1:1-Beziehung zu der Mannschaft des Spielers und dann weiter entlang einer 1:1-Beziehung zu dem Trainer dieser Mannschaft navigiert. Von diesem Trainer wird der Name ermittelt, heißt dieser Müller, so ist die Restriktionsbedingung erfüllt, der entsprechende Spieler wird in die Ergebnismenge aufgenommen.

Äquivalent zu der Objektnavigation in Abbildung 3 wird die From-Anweisung erweitert. Für jede neue Navigation wird ein neuer Teil hinzugefügt. Dementsprechend kommt in der From-Anweisung zuerst ein zweiter Teil hinzu, der zu jedem Spieler `s` die zu ihm gehörigen Mannschaften ermittelt. Die Menge der Mannschaften hat bei 1:1-Beziehungen und konsistenter Datenbank die Kardinalität 1. Die jeweils aktuelle und bei 1:1-Beziehungen einzige Ausprägung dieser Menge wird als `nav1` bezeichnet. Der Bezeichner `nav1` steht somit für die Navigation `s.mannschaft`. Für die nächste Navigation von der Mannschaft zu dem Trainer, muss in der From-Anweisung ein dritter Teil ergänzt werden. Dieser ermittelt zu jeder Mannschaft des Spielers `s` den zugehörigen Trainer. Da `nav1` gerade die Mannschaft des Spielers `s` repräsentiert, wird zu jeder Ausprägung von `nav1` der zugehörige Trainer ermittelt, der dann als `nav2` bezeichnet wird.

Würde die Anfrage mehr als nur eine zweifache Navigation verwenden, so müsste für jede weitere Navigation entsprechend der oben beschriebenen Vorgehensweise ein neuer Teil zu der From-Anweisung hinzugefügt werden. Das Verfahren garantiert somit, dass beliebig lange Navigationen nach SQL umsetzbar sind.

**Transformation von OQL nach SQL**  
Objektnavigation in der Where-Anweisung bei 1:1 Beziehungen

**Aktion:** Ermittle alle Spieler, deren Trainer den Namen "Müller" hat.

**OQL-Anfrage:**

```
select *
from   Spieler s
where  s.mannschaft.trainer.name like "Müller"
```

**Zwischenschritt:**

```
SELECT s.OID, s.NAME, s.VORNAME, s.TORE, s.FOULS,
       s.MANNSCHAFTSID
FROM   SPIELER s,
       (SELECT * FROM MANNSCHAFT
        WHERE OID=s.MANNSCHAFTSID) nav1,
       (SELECT * FROM TRAINER,
        WHERE OID=nav1.TRAINERID) nav2
WHERE  nav2.NAME LIKE 'Müller'
```

**Generierte SQL-Anfrage:**

```
SELECT s.OID, s.NAME, s.VORNAME, s.TORE, s.FOULS,
       s.MANNSCHAFTID
FROM   SPIELER s, MANNSCHAFT nav1, TRAINER nav2
WHERE  (s.MANNSCHAFTID=nav1.OID) AND
       (nav1.TRAINERID=nav2.OID) AND
       (nav2.NAME LIKE 'Müller')
```

Abbildung 4: Zweifache Objektnavigation in der Where-Anweisung

Als Resultat des oben angegebenen Verfahrens erhält man die im Zwischenschritt angegebene SQL-Anfrage. Die Umwandlung der Anfrage des Zwischenschrittes zu der generierten SQL-Anfrage geschieht analog zum letzten Beispiel: Anstatt zuerst die Restriktionen durchzuführen und dann die kartesischen Produkte zu bilden, werden zuerst die kartesischen Produkte gebildet und danach die Restriktionen durchgeführt.

Auch bei diesem Beispiel wird ein weiterer Sonderfall erkennbar: Die OQL-Anfrage verwendet den speziellen \*-Operator in der Select-Anweisung. Dies hat zur Folge, dass der Ergebnis-Typ genau dem Typ der From-Anweisung der OQL-Anfrage entspricht, also keine konkrete Projektion vorgenommen wird. In der generierten SQL-Anfrage sind weitere Einträge in die From-Anweisung hinzugekommen, d.h. die Größe der Tupel hat sich verändert und somit auch ihr Typ. Daher kann der \*-Operator, der in SQL eigentlich gleiche Bedeutung wie in OQL hat, nicht in der Select-Anweisung verwendet werden. Anstelle des \*-Operators, sind die einzelnen Spalten, wie in Abbildung 4 gezeigt, explizit aufzuzählen.

### **4.3. Navigation in der From-Anweisung**

Abbildung 5 zeigt ebenfalls eine zweifache Objektnavigation, diesmal jedoch in einer From-Anweisung. In den bisherigen zwei Beispielen wurden lediglich 1:1-Beziehungen betrachtet. In diesem Beispiel findet zuerst eine Objektnavigation entlang einer 1:1-Beziehung von der

aktuellen Trainerausprägung  $t$  zu dessen Mannschaft statt. Von dieser Mannschaft wird weiter entlang einer 1:N-Beziehung zu der Menge der Spieler dieser Mannschaft navigiert. Die einzelnen Spielerausprägungen dieser Menge werden mit  $s$  bezeichnet.

### Transformation von OQL nach SQL

Objektnavigation in der From-Anweisung

<b>Aktion:</b>	Ermittle die Namen der Trainer, die mindestens einen Spieler mit dem Namen Markus trainieren.
<b>OQL-Anfrage:</b>	<pre>select t.name from  Trainer t, t.mannschaft.spieler s where s.name like "Markus"</pre>
<b>Zwischen- schritt:</b>	<pre>SELECT t.NAME FROM  TRAINER t,       (SELECT * FROM MANNSCHAFT        WHERE TRAINERID=t.OID) nav1,       (SELECT * FROM SPIELER,        WHERE MANNSCHAFTSID=nav1.OID) s WHERE s.NAME LIKE 'Markus'</pre>
<b>Generierte SQL-Anfrage:</b>	<pre>SELECT t.NAME FROM  TRAINER t, MANNSCHAFT nav1, SPIELER s WHERE (t.OID = nav1.TRAINERID) AND       (nav1.OID = s.MANNSCHAFTID) AND       (s.NAME LIKE 'Markus')</pre>

Abbildung 5: Objektnavigation in der From-Anweisung

Entsprechend den bisherigen Beispielen wird für jede Navigation ein neuer Teil in die zu erzeugende From-Anweisung hinzugefügt, zusätzlich dürfen die Objektnavigationen aus der From-Anweisung der OQL-Anfrage nicht übernommen werden.

Vergleicht man das Beispiel aus Abbildung 5 mit dem Beispiel aus Abbildung 4, so stellt man fest, dass es für die Generierung der SQL-Anfrage unerheblich ist, ob es sich bei der Navigation um eine 1:1- oder eine 1:N-Beziehung handelt, denn 1:1-Beziehungen werden als Sonderfall von 1:N-Beziehungen behandelt. Nach dem bereits bekannten Vorgehen erhält man die im Zwischenschritt gezeigte SQL-Anfrage.

Die Umwandlung von der Anfrage des Zwischenschrittes zu der generierten SQL-Anfrage geschieht wiederum analog: Anstatt zuerst die Restriktionen durchzuführen und dann die kartesischen Produkte zu bilden, werden zuerst die kartesischen Produkte gebildet und dann die Restriktionen durchgeführt.

Aber auch dieses Beispiel zeigt einen weiteren Sonderfall auf: In der OQL-Anfrage wird jede Ausprägung aus der Menge der Spieler, die durch die zweifache Objektnavigation  $t.mannschaft.spieler$  bestimmt wird, durch  $s$  bezeichnet. Für diese Navigation darf kein

automatisch generierter Bezeichner, wie etwa `nav2`, verwendet werden. Vielmehr ist der explizit angegebene Bezeichner `s` zu verwenden, wie in Abbildung 5 gezeigt.

#### **4.4. Navigation in eingeschachtelter Select From Where-Anweisung**

Die bisherigen Beispiele waren alle von relativ einfacher Natur. Objektnavigationen traten nur gesondert in einer der Select-, From- oder Where-Anweisungen auf. Das Beispiel in Abbildung 6 zeigt wie mehrfach, an unterschiedlichen Stellen auftretende Objektnavigation umgesetzt werden. Als Erweiterung findet auch eine Objektnavigation innerhalb einer eingeschachtelten Select From Where-Anweisung sowie einer Order-by-Anweisung statt.

Zuerst wird die From-Anweisung der OQL-Anfrage aus Abbildung 6 betrachtet. Diese enthält keine Objektnavigation. Es reicht also, für die zu erzeugende From-Anweisung den Klassennamen durch den Relationen- bzw. Sichtennamen zu ersetzen. Im Anschluss daran wird die äußerste Where-Anweisung bearbeitet. Die erste Navigation von der aktuellen Trainerausprägung zur Mannschaft wird wie bisher umgesetzt: Die äußerste From-Anweisung wird um einen weiteren Teil ergänzt, dessen aktuelle Ausprägung mit `nav1` bezeichnet wird. Bei der weiteren Abarbeitung der äußersten Where-Anweisung trifft man auf die innerhalb der Exists-Anweisung eingeschachtelte Select From Where Anweisung. In deren From-Anweisung findet eine zweifache Navigation statt, von der aktuellen Trainerausprägung zur Mannschaft des Trainers und von dieser Mannschaft zu deren Spielern. Die erste Navigation trat oben bereits auf, ihre aktuelle Ausprägung wird durch den Bezeichner `nav1` beschrieben. Es reicht also von der aktuellen Mannschaftsausprägung `nav1` zu der Menge der Spieler zu navigieren. Diese Navigation wird ebenfalls wie bisher umgesetzt: Die eingeschachtelte From-Anweisung wird um einen Teil ergänzt, der die Spieler der aktuellen Mannschaft `nav1` bestimmt. Die jeweils aktuelle Spielerausprägung wird entsprechend der OQL-Anfrage mit `s` bezeichnet. Die Navigation in der From-Anweisung wird, wie im letzten Beispiel, weggelassen bzw. nicht in die zu generierende Anfrage übernommen. Im Anschluss daran wird die eingeschachtelte Where-Anweisung bearbeitet; diese kann auf Grund ihrer Einfachheit nahezu direkt nach SQL übernommen werden. Nach Abarbeitung dieser Where-Anweisung ist die eingeschachtelte Select From Where-Anweisung umgesetzt worden und damit auch die äußerste Where-Anweisung. Es folgt die Abarbeitung der order by-Anweisung. Diese enthält die Objektnavigation `t.mannschaft` die bereits durch den Bezeichner `nav1` ausgedrückt wurde. Es reicht also, diese Navigation durch den Bezeichner `nav1` zu ersetzen. Nach bereits bekannter Anpassung der äußersten Select-Anweisung erhält man dann den in Abbildung 6 angegebenen Zwischenschritt.

## Transformation von OQL nach SQL

### Objektnavigation in eingeschachtelter Select From Where-Anweisung

**Aktion:** Ermittle die Trainernamen und deren Mannschaft, aufsteigend sortiert nach dem Mannschaftskapital, die eine Mannschaft mit dem Präfix "FC" trainieren und deren Mannschaft mindestens einen Spieler mit dem Namen Markus hat.

**OQL-Anfrage:**

```
select t.name, t.mannschaft
from Trainer t
where (t.mannschaft.name like "FC*") and
exists(select *
        from t.mannschaft.spieler s
        where s.name like "Markus")
order by t.mannschaft.kapital asc
```

**Zwischenschritt:**

```
SELECT t.NAME, nav1.OID, nav1.NAME, nav1.TABELLENPLATZ,
nav1.KAPITAL, nav1.TRAINERID
FROM TRAINER t, (SELECT * FROM MANNSCHAFT WHERE
t.OID=TRAINERID) nav1
WHERE (nav1.NAME LIKE 'FC%') AND
EXISTS(SELECT s.OID
        FROM (SELECT * FROM SPIELER
              WHERE nav1.OID=MANNSCHAFTID) s
        WHERE s.NAME LIKE 'Markus'))
ORDER BY nav1.KAPITAL ASC
```

**Generierte SQL-Anfrage:**

```
SELECT t.NAME, nav1.OID, nav1.NAME, nav1.TABELLENPLATZ,
nav1.KAPITAL, nav1.TRAINERID
FROM TRAINER t, MANNSCHAFT nav1
WHERE (t.OID=nav1.TRAINERID) AND
(nav1.NAME LIKE 'FC%')AND
EXISTS(SELECT s.OID
        FROM SPIELER s
        WHERE (t.OID=nav1.TRAINERID) AND
              (nav1.OID=s.MANNSCHAFTID) AND
              (s.NAME LIKE 'Markus'))
ORDER BY nav1.KAPITAL ASC
```

Abbildung 6: Navigation in eingeschachtelter Select From Where-Anweisung

Von dem Zwischenschritt zu der generierten SQL-Anfrage gelangt man wie bisher, indem zuerst die kartesischen Produkte und dann die Restriktionen gebildet werden.

## **4.5. Zusammenfassung: Objektnavigation**

Die letzten vier Beispiele haben exemplarisch gezeigt wie die Objektnavigation umgesetzt werden kann. Hierbei wurde in 2 Phasen, nach folgendem Schema, vorgegangen:

### *1. Phase:*

Für jede Objektnavigation, egal in welchem Teil einer Select From Where Order by – Anweisung sie auch auftritt, wird der From-Teil dieser aktuell betrachteten Anweisung um eine weitere Komponente erweitert. Es wird also ein kartesisches Produkt gebildet. Die hinzugefügte Komponente stellt eine komplette Select From Where-Anweisung dar, welche für das aktuelle Objekt (=aktuelle Ausprägung) der Quellklasse die zugehörigen Objekte der Zielklasse ermittelt. Dieser eingefügten Komponente wird, falls vorhanden, ein expliziter Bezeichner zugeordnet, ansonsten wird automatisch ein, innerhalb der Anfrage, eindeutiger Bezeichner erstellt. Tritt die Objektnavigation in der From-Anweisung auf, so ist sie zu eliminieren, andernfalls ist sie durch den erstellten bzw. vorhandenen Bezeichner zu ersetzen. Enthält eine Select-Anweisung den \*-Operator, so muss dieser durch eine explizite Auszählung der Ergebnisspalten ersetzt werden. Enthält eine Select-Anweisung als Projektion ein Objekt, so müssen statt dem Objekt alle Spalten, die das Objekt repräsentieren, aufgelistet werden. Nachdem Klassen- durch Tabellennamen sowie Attribut- durch Spaltennamen ersetzt wurden, erhält man die in den Beispielen angegebenen Zwischenschritte als Ergebnis der ersten Phase.

### *2. Phase:*

Nach der 1. Phase enthalten möglicherweise manche From-Anweisungen eingeschachtelte Select From Where-Anweisungen. Diese werden in der zweiten Phase aufgelöst. Dabei werden, wie in den bisherigen Beispielen, zuerst die kartesischen Produkte gebildet und dann die Restriktionen gebildet.

Hierzu wird für jede eingeschachtelte Anweisung wie folgt vorgegangen: Die From-Anweisung der eingeschachtelten Anweisung wird in die From-Anweisung der sie umgebenden Select From Where-Anweisung übernommen. Als Bezeichner wird der Bezeichner der eingeschachtelten Anweisung verwendet. Ferner wird die Where-Anweisung der eingeschachtelten Anweisung in die Where-Anweisung der sie umgebenden Select From Where-Anweisung übernommen. Der Select-Teil der eingeschachtelten Anweisung wird verworfen. Diese Umwandlung ist nur möglich, wenn entweder beide oder keine der Select-Anweisungen das Schlüsselwort `distinct` verwenden.

In den obigen vier Beispielen wird die zweite Phase durch den Vorgang von dem Zwischenschritt zu der generierten SQL-Anfrage charakterisiert.

#### **4.6. Ablauf der Auswertung**

Bisher wurde beschrieben, wie OQL-Anfragen bestimmter Ausprägung in semantisch äquivalente SQL-Anfragen übersetzt werden können. Für die gesamte Auswertung einer OQL-Anfrage stellt diese Umwandlung aber nur einen kleinen Aufgabenteil dar, wie Abbildung 7 zeigt.



Abbildung 7: Ablauf der Auswertung einer OQL-Suchanfrage

Die von dem Anwender oder Software-Entwickler erstellte OQL-Anfrage, in Form einer Zeichenkette, wird in eine einzige, gleichbedeutende SQL-Anfrage übersetzt, welche von dem relationalen Datenbanksystem ausgewertet wird. Das Ergebnis dieser Anfrage ist eine Menge von Tupeln, wobei die Komponenten der Tupel relationalen Datentypen entsprechen.

Die elementaren, relationalen Datentypen müssen in korrespondierende ODMG bzw. Java-Datentypen umgewandelt werden. Außerdem müssen aus den Tupeln oder aus Teilen eines Tupels fachliche Objekte erstellt werden; natürlich nur sofern diese Objekte repräsentieren.

Je nach Art der OQL-Anfrage werden, entsprechend der Sprachbeschreibung von OQL (siehe [Catt98]), die Ergebniselemente in einer Menge, einer Multimenge oder einem Feld gespeichert.

Durch die beschriebene Vorgehensweise wird die relationale Datenhaltung für persistente Objekte komplett vor dem Benutzer verborgen: Er formuliert Anfragen auf dem Objektmodell in Form von OQL-Anfragen und bekommt das Ergebnis wieder auf der Ebene der Objekte in Form von ODMG/Java Datentypen.

## 5. Interpretation von OQL

Es gibt aber auch OQL-Anfragen, die nicht in semantisch äquivalente SQL-Anfragen übersetzt werden können. Sie sind daher nicht direkt von einem relationalen Datenbanksystem auswertbar. Vielmehr ist ein eigener *OQL-Auswerter* (im weiteren *OQL-Interpreter* genannt) notwendig, der in zwei Phasen agiert.

In der *ersten Phase* fordert er die für die Auswertung der OQL-Anfrage benötigten Daten von dem relationalen Datenbanksystem an. Dies erfolgt mit Hilfe von sehr einfachen SQL-Anfragen. Aus den ermittelten Daten werden, wie Abbildung 8 zeigt, die sie repräsentierenden Objekte gebildet, die für die weitere Auswertung notwendig sind. Außerdem müssen auch die elementaren SQL-Datentypen, die Attribute repräsentieren, in entsprechende ODMG-Datentypen umgewandelt werden.

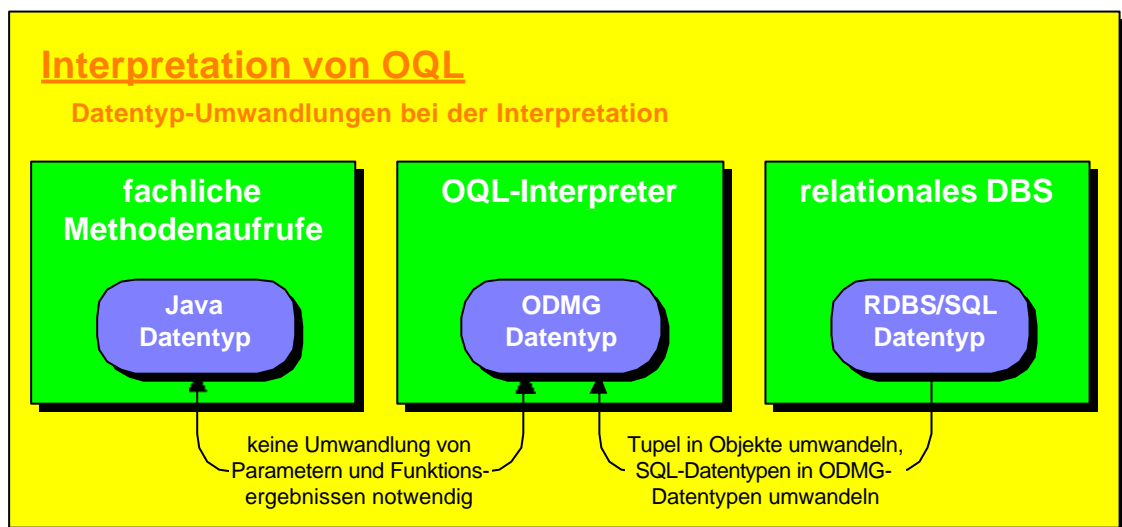


Abbildung 8: Umwandlung von Datentypen bei der Interpretation

In der *zweiten Phase* wertet der OQL-Interpreter die Anfrage anhand der ermittelten Objekte aus. Hierzu werden evtl. die in der Anfrage vorhandenen fachlichen Methoden- und Konstruktorenaufrufe per Java - Reflection ausgewertet. Das von der ODMG definierte Java Language Binding (siehe [Catt98]) sorgt dafür, dass sich die Sprache OQL anstandslos in das Datenmodell von Java einfügt. Parameter für fachliche Methodenaufrufe sowie Ergebnisse von fachlichen Methodenaufrufen müssen, wie Abbildung 8 zeigt, nicht umgewandelt werden.

Wie leicht ersichtlich ist, scheidet durch diese 2-Phasen Vorgehensweise jegliche Low-Level Optimierung der OQL-Anfrage aus. Allerdings kann eine auf der Semantik beruhende High-Level Optimierung vorgenommen werden. Hierzu ist denkbar, dass die OQL-Anfrage in eine für die Optimierung geeignetere Darstellungsform gebracht wird, zum Beispiel in die eines Ablaufgraphen.

Im folgenden wird beschrieben, wie die Objektnavigation bei der Interpretation von OQL-Anfragen umgesetzt werden kann.



Abbildung 9: Konkrete Objektnavigation bei der Interpretation

Für jede durchzuführende Objektnavigation ist jedes Objekt der Quellklasse (bzw. Tupel) bereits bekannt. Abbildung 9 zeigt genau ein solches Objekt der Quellklasse Mannschaft. Mit dessen Hilfe kann eine für dieses Objekt spezifische SQL-Anfrage generiert werden, die das zugehörige Objekt bzw. bei zu N-Beziehungen die zugehörigen Objekte der Zielklasse ermittelt. Im Beispiel wird für ein spezielles Mannschaftsobjekt der zugehörige Trainer bestimmt.

Dieses Verfahren ist jedoch extrem zeitaufwendig, da für jede Navigation, von einem konkreten Objekt der Quellklasse ausgehend, eine SQL-Anfrage erzeugt und an das Datenbanksystem abgesetzt werden muss, um das entsprechende Objekt oder die entsprechenden Objekte der Zielklasse zu ermitteln. Will man zum Beispiel zu jeder Mannschaft den zugehörigen Trainer bestimmen und nimmt man an, dass in der Datenbank insgesamt 1000 Mannschaften gespeichert sind, so müssten bei obigem Verfahren insgesamt 1001 SQL-Anfragen abgesetzt werden. Eine SQL-Anfrage wird benötigt um die 1000 Mannschaften zu bestimmen, und für jede der 1000 Mannschaften eine weitere, um den entsprechenden Trainer zu ermitteln (wie in Abbildung 9 exemplarisch für ein Objekt gezeigt). Dies ist in der Praxis nicht machbar.

Die Anzahl der zu erzeugenden SQL-Anfragen muss drastisch reduziert werden: Bereits bei der Typüberprüfung wird ermittelt, von welchen Quellklassen zu welchen Zielklassen navigiert wird. In der 1. Phase der Interpretation, werden zu allen Objekten der Quellklassen auch gleich alle Objekte der Zielklassen ermittelt. Hierfür werden insgesamt lediglich zwei SQL-Anfragen benötigt. Im Anschluss daran werden im Speicher den jeweiligen Objekten einer Quellklasse ihre zugehörigen Objekte der Zielklasse in Form einer Referenz oder einer Menge von Referenzen zugeordnet. Für eine Navigation von einem Quellobjekt zu dem zugehörigen Zielobjekt bzw. den zugehörigen Zielobjekten muss also nur noch der entsprechenden Referenz bzw. den entsprechenden Referenzen gefolgt werden. Es muss keine weitere SQL-Anfrage erzeugt werden.

Der Vorteil dieser Vorgehensweise ist, dass die Anzahl der zu erzeugenden SQL-Anfragen drastisch reduziert werden kann – im Beispiel von 1001 auf 2 – was einen sehr positiven Einfluss auf das Laufzeitverhalten hat. Der Speicherverbrauch bleibt gegenüber dem ersten Verfahren nahezu unverändert, da sowieso alle Objekte für die Interpretation im Speicher gehalten werden müssen. Die Attribute, welche die Beziehungen (als Referenz) speichern, sind ohne Anwendung dieser Optimierung leer, bei Verwendung der Optimierung enthalten sie die entsprechenden Verweise.

## **6. Zusammenfassung**

Der Beitrag hat gezeigt, wie die bei objektorientierten Datenbanksystemen beliebte Anfragesprache OQL des ODMG Standards bei relationaler Datenhaltung umgesetzt werden kann. Als Resultat ist damit das Problem gelöst, für Objektmodelle mit relationaler Datenhaltung, eine sowohl effiziente als auch mächtige, objektorientierte Anfragesprache zu finden. Die meisten, in der Praxis benötigten OQL-Anfragen können durch die direkte Umsetzung in gleichbedeutende SQL-Anfragen extrem effizient ausgeführt werden. Unabhängig von der relationalen Datenhaltung findet die Formulierung der Anfrage selbst, wie auch dessen Ergebnis, auf der Ebene der Objekte statt. Das von der ODMG definierte Language Binding sorgt dabei für die problemlose Einbettung von OQL in die Programmiersprache bzw. deren Datenmodell.

Somit steht mit OQL, für alte Systeme mit relationaler Datenhaltung, eine der modernsten Anfragesprachen zur Verfügung.

## **7. Literatur**

- [Catt98] Cattell R.G.G., Barry D.K.: *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publishers, San Francisco 1998.
- [ISO] ISO/IEC 9075:1992(E): *Information technology - Database languages - SQL*, 3. Aufl. 1992
- [Wan96] Wanner G.: *Datenbanksysteme und Objektverteilung in kommerziellen Systemen*, Zeitschrift it-Management, 11/12 1996, S.18-26
- [Wan97] Wanner G.: *Transformation einer objektorientierten Anfragesprache in SQL-Anweisungen*, Veröffentlichung in UNIX OPEN 9/97, S.54-58