

Hauptseminar | Wintersemester 2004/05

Semantik der UML 2.0

Manfred Broy

Seminarausarbeitung

Betreuer: Stefan Wagner

Referentin: Han Jiang

A Methodological Approach for Object-Relational Database
Design using UML

Datum des Referats: 26. November.2004

1. Datenbank Geschichte	3
1.1 Hierarchisches Datenbankmodell.....	3
1.2 Netzwerk-Modell.....	3
1.3 Relationales Datenbankmodell.....	4
1.4 Objekt-Relationales Datenbankmodell.....	4
2. UML Einführung	5
2.1 Klassendiagramm.....	6
2.1.1 Klassen.....	6
2.1.2 Attribute.....	7
2.1.3 Operation.....	8
2.1.4 Zusicherung, Merkmal und Notizen.....	8
2.1.5 Assoziationen.....	8
2.1.6 Aggregation.....	9
2.1.7 Komposition.....	9
2.1.8 Vererbung.....	10
2.1.9 Abhängig.....	10
2.2 UML Profile.....	11
3. Objekt-relationales Datenbank-Design mit der UML	11
3.1 Die objekt-relationale Modelle: SQL:1999 und Oracle8i.....	12
3.2 Objekt-relationale Erweiterung für UML.....	13
3.3 Methodologie für objekt-relationales Datenbank-Design.....	14
3.4 Transformationsrichtlinien.....	14
3.5 Semantik der UML.....	17
3.6 Vergleich zwischen Semantik von SQL und der Superstructure.....	18
4. Zusammenfassung	19
5. Literaturen	20
Anhang A: Stereotypen von SQL:1999.....	21
Anhang B: Stereotypen von Oracle8i.....	22

Die meisten Datenbanken werden mittels ein konzeptuelles Modell, z.B. E/R Diagramm, entworfen. Das Modell berücksichtigt keine andere Sichten des Systems. Die neue objekt-orientierte Sprache (z.B. UML) erlaubt, dass das ganze System in eine gleiche Art und Weise modelliert wird. Außerdem ist die UML eine erweiterbare Sprache und erlaubt beliebige nötige Einsetzen von neue Stereotypen für spezielle Applikationen.

Heutzutage sind die relationalen Datenbanksysteme marktbeherrschend. Aber neue Applikationen bedürfen, dass die komplexe Objekte auch in Datenbanken gespeichert werden sollen. Das objekt-relationale Modell ist geeigneter als relationales Modell für dieses Bedürfnis. In diesem Referat versuchen wir ein objekt-relationales Datenmodell mittels UML zu entwerfen.

1. Datenbank Geschichte

1.1 Hierarchisches Datenbankmodell

Das Hierarchische Datenbankmodell ist das älteste Datenbankmodell. Es bildet die Realwelt durch einen hierarchischen Baum ab. Das entspricht einem Eltern-Kind-Prinzip. Jeder Satz (Record) hat genau einen Vorgänger und genau ein Satz bildet die Wurzel. Ein Nachteil ist, dass ein einzelner Dateneintrag mehrfach gespeichert werden kann. Dies verursacht Speicherverschwendung. Ein anderer Nachteil ist, dass eine Verknüpfung über mehrere Ebenen nicht möglich ist. Dies verursacht lange Suchzeiten. Das Modell ging aus den Informationsmanagementsystemen (IMS) in den 1950er und 1960er Jahren hervor und wurde von vielen Banken und Versicherungsunternehmen eingesetzt. Dort findet man hierarchische Datenbanken z. T. noch heute. [1]

1.2 Netzwerk-Modell

Das Netzwerkmodell einer Datenbank wurde 1969 von der Data Base Task Group (DBTG) von CODASYL (Conference on Data Systems Languages) formuliert. Im Gegensatz zu hierarchischen Datenbanken können die Datensätze in Netzwerkdatenbanken auf mehreren Wegen verknüpft sein. Ein Vorteil dabei ist eine schnellere Suchzeit gegenüber hierarchischen Datenbanken. Aber die Struktur wird sehr schnell ziemlich undurchsichtig. Ein Beispiel für das Netzwerk-Modell ist: IDS (Integrated Data Store) von General Electric.[1]

1.3 Relationales Datenbankmodell

Relationale Datenbanken sind heute am weitesten verbreitet. Das relationale Datenbankmodell wurde 1970 bei IBM entworfen.[1] Es ist eine Datenbank, die auf dem Entity-Relationship-Modell basiert. Die Daten werden dabei in Form von zweidimensionalen Tabellen verwaltet, die über Schlüssel miteinander verknüpft werden können. Diese Verknüpfungen werden Relationen genannt. Die meisten eingesetzten Datenbankverwaltungssysteme sind relationale Datenbankverwaltungssysteme, z.B. Microsoft Access, Oracle, Mysql. Der Hauptvorteil des relationalen Datenbankmodells ist, dass die Struktur der Datenbank verändert werden kann, ohne dass deswegen Anwendungen geändert werden müssen, die sich auf die ältere Struktur gründeten. Ein weiterer Vorteil des relationalen Modells ist, dass man beliebig viele Sichten (views) oder virtuelle Tabellen (virtual tables) der Daten mit unterschiedlichster logischer Struktur schaffen kann, indem verschiedene Tabellen bzw. Teile von Tabellen kombiniert werden. Dazu muss die Datenbank physisch nicht verändert werden.[1]

1.4 Objekt-Relationales Datenbankmodell

Heutzutage gibt es immer mehr komplexe Objekte mit komplexen Beziehungen. Um solche Objekte und Beziehungen in einem relationalen Modell zu repräsentieren, müssen sie in viele Tabellen unterteilt werden. Das bedeutet, es werden mehrere Verknüpfungen gebraucht, um ein Objekt wiederzubekommen. Wenn die Tabellen zu tief verschachtelt sind, ist die Leistung wesentlich geringer. Eine neue Generation von Datenbanken kann solche Probleme lösen. Das ist das objekt-relationale Datenbankmodell. Die Erweiterungen beziehen sich im wesentlichen auf folgende Aspekte:

Große Objekt: Hierbei handelt es sich um Datentypen, die es erlauben, auch sehr große Attributwerte für z.B. Multimedia-Daten zu speichern. Die Größe kann bis zu einigen Gigabytes betragen.

Mengenwertige Attribute: Im relationalen Datenbankmodell dürfen die Attribute nur atomare Typen haben z.B. int, String, Date. Im objekt-relationalen Datenbankmodell hingegen können auch mengenwertige Attribute enthalten sein.

Geschachtelte Relationen: Bei geschachtelten Relationen geht man noch einen Schritt weiter als bei mengenwertigen Attributen und erlaubt Attribute, die selbst wiederum

Relationen sind.

Typedeklarationen: Objekt-relationale Datenbanksysteme unterstützen die Definition von anwendungsspezifischen Typen – oft user-defined types (UDTs) genannt. Dadurch kann man dann komplexe Objektstrukturen aufbauen.

Referenzen: Attribute können direkte Referenzen auf Tupel/Objekte (derselben oder anderer Relationen) als Werte haben.

Objektidentität: Referenzen setzen natürlich voraus, dass man Tupel/Objekte anhand einer Objektidentität eindeutig identifizieren kann.

Vererbung: ein komplex strukturierten Typen können von einem Obertyp erben.

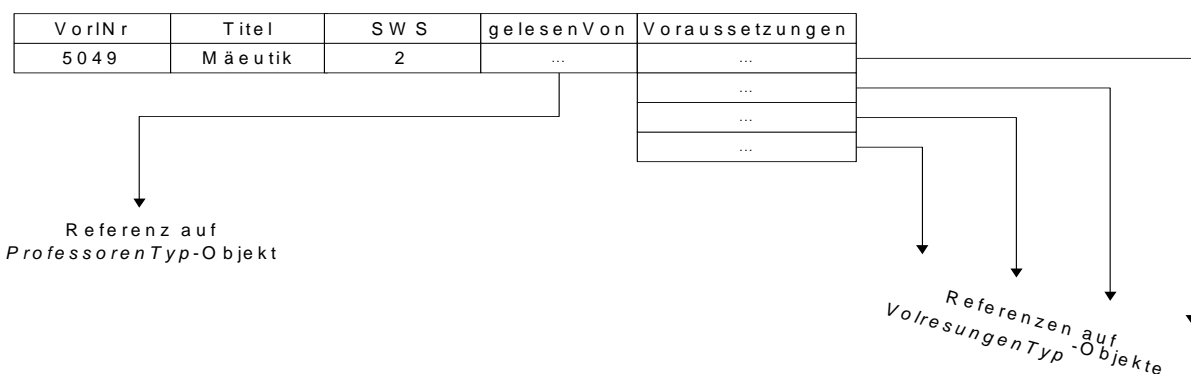


Bild 1. Die Relation Vorlesung[2]

Hier ist ein sehr einfaches Beispiel. Die Tabelle Vorlesung besteht aus fünf Spalten. Die *VolNr*, *Titel* und *SWS* haben atomare Typen. Das Attribut *gelesenVon* ist eine Referenz, die an einem Professor verweist. Das Attribut *Voraussetzungen* ist ein mengenwertiges Attribut. Das enthält eine Menge von Referenzen, die wieder Tupel von Vorlesungen referenzieren.

2. UML Einführung

UML ist eine Abkürzung für Unified Modeling Language. Es ist eine Sprache und Notation zur Spezifikation, Visualisierung, Konstruktion und Dokumentation von Modellen für Softwaresysteme. Sie ist jedoch bewusst keine Methode. Sie ist lediglich ein Satz von Notationen zur Formung einer allgemeinen Sprache zur Softwareentwicklung. Eine Methode beinhaltet Empfehlungen zur Vorgehensweise bei der Entwicklung.

Wozu wird UML gebraucht? Software wird zunehmend komplexer, z.B. Windows 2000 hat bis 40 Millionen Zeilen Programmcode. Einige Datenbankverwaltungssysteme haben

mehrere Gigabytes Programmcode. Es ist unmöglich für einen Programmierer allein den ganze Code zu verstehen und zu verwalten. Außerdem ist es schwer für Entwickler Code zu verstehen, den er nicht geschrieben hat.

Wir brauchen eine einfache Repräsentation für komplexe Systeme. Mit UML kann man komplexe Systeme graphisch darstellen, um das System besser zu verstehen und zu verwalten.

Es gibt bei der UML mehrerer Gliederungskriterien. Nach Diagrammtypen hat UML folgende Diagramme: Anwendungsfalldiagramm, Klassendiagramm Aktivitätsdiagramm, Kollaborationsdiagramm, Sequenzdiagramm, Zustandsdiagramm usw.

2.1 Klassendiagramm

Ein Klassendiagramm zeigt eine Menge statischer Modellelemente, vor allem Klassen und ihre Beziehungen. Ein Klassendiagramm dient der Darstellung der statischen Entwurfssicht eines Systems. [6]

2.1.1 Klassen

Eine Klasse ist eine Menge von Objekten, in der die Eigenschaften(Attribute), Operationen und die Semantik der Objekte definiert werden. Klassen werden durch Rechtecke dargestellt, die den Namen der Klasse und/oder die Attribute und Operationen der Klasse enthalten. Klassenname, Attribute und Operationen werden durch eine horizontale Linie getrennt. Der Klassenname steht im Singular und beginnt mit einem Großbuchstaben.

Eine Metaklasse ist eine Klasse, deren Instanzen wiederum Klassen sind. Dieses Konzept existiert nur in einigen objektorientierten Sprachen (z.B. in Smalltalk).

Eine Parametrisierbare Klasse ist eine mit generischen formalen Parametern versehene Schablone, mit der gewöhnliche (d.h. nicht generische) Klassen erzeugt werden können. Die generischen Parameter dienen als Stellvertreter für die aktuellen Parameter, die Klassen oder einfache Datentype repräsentieren. Parametrisierbare Klassen werden wie Klassen dargestellt, sie erhalten aber zusätzlich in der rechten oberen Ecke in einem gestrichelten Rechteck die notwendigen Parameter. Eine Klasse, die durch eine parametrisierbare Klasse erzeugt wird, muss mit einer Verfeinerungsbeziehung, die den Stereotyp "bind" erhält gekennzeichnet werden.

Schnittstellen beinhalten eine Menge von Signaturen für Operationen, die nicht implementiert werden dürfen. Schnittstellen sind mit dem Schlüsselwort „interface“ mit doppelten Spitzklammern gekennzeichnet. Gewöhnliche Klassen, die eine Schnittstelle implementieren wollen, müssen alle in der zugehörigen Schnittstellenklasse definierten Operationen implementieren. Eine gewöhnliche Klasse kann mehrere Schnittstellen implementieren und darüber hinaus weitere Eigenschaften enthalten.

Eine abstrakte Klasse bildet die Grundlage für weitere Unterklassen. Sie wurde bewusst unvollständig gehalten. Von ihr können keine konkreten Instanzen erzeugt werden. Eine abstrakte Klasse wird wie eine normale Klasse dargestellt. Unter dem Klassennamen steht das Merkmal *abstract*.

2.1.2 Attribute

Ein Attribut ist ein Datenelement, das in jedem Objekt einer Klasse enthalten ist und von jedem Objekt mit einem individuellen Wert (entspricht Instanzvariablen) oder mit gleichem Wert (entspricht Klassenvariablen) repräsentiert wird. Jedes Attribut wird mindestens durch seinen Namen beschrieben. Zusätzlich können durch Datentyp bzw. eine Klasse sowie ein Initialwert, Merkmal und Zusicherungen definiert werden. Mit Angabe von Eigenschaftswerten (Merkmal) können weitere besondere Eigenschaften beschrieben werden. Mit Zusicherungen kann man z.B. den Wertebereich des Attributes einschränken. Abgeleitete Attribute werden innerhalb eines Objektes automatisch berechnet. Die Berechnungsvorschrift wird in Form einer Zusicherung angegeben. Abgeleitete Attribute werden durch einen vorangestellten Schrägstrich „/“ markiert.

Eine weitere Ausprägung von Attributen sind Klassenattribute. Sie gehören nicht zu einem einzelnen Objekt, sondern zu einer Klasse. Alle Objekte dieser Klasse können ein solches Klassenattribut benutzen.

Es gibt, je nach Programmiersprache, die Möglichkeit die Sichtbarkeit der Attribute nach außen hin einzuschränken. Dies geschieht mit Hilfe der Sichtbarkeitskennzeichen:

- *public*: verfügbar für alle Klassen des Systems
- *protected*: verfügbar für Objekte der eigenen und aller abgeleiteten Klassen
- *private*: verfügbar nur für Objekte dieser Klasse
- *Package*: verfügbar nur für Klassen im gleichen.

Sichtbarkeitsangaben wie *public*, *protected*, *private* und *package* werden mit „+“, „#“, „-“, „~“ gekennzeichnet.

2.1.3 Operationen

Operationen sind Dienstleistungen, die von einem Objekt angefordert werden können. Sie werden beschrieben durch ihre Signatur (Operationsname, Parameter, Rückgabewert) sowie Merkmal und Zusicherung. Die Markierungen in Operationen sind fast gleich wie bei Attributen. Eine Besonderheit sind abstrakte Operationen. Sie sind solche, die nur durch ihre Signatur repräsentiert werden und deren Implementierung erst in einer Unterklasse stattfindet. Abstrakte Operationen werden wahlweise kursiv gesetzt und erhalten den Eigenschaftswert *{abstract}*.

2.1.4 Zusicherung, Merkmal und Notizen

Eine Zusicherung beschreibt eine Bedingung oder ein Integritätsregel.

Merkmale fügen weitere Eigenschaften an Modellelemente hinzu z.B. ob eine Methode abstrakt ist.

Notizen sind Kommentare zu einem Diagramm oder beliebigen Modellelementen, ohne semantische Bedeutung. Notizen werden durch Rechtecke mit einem Eselsohr dargestellt, die einen Text enthalten und durch einer gestrichelten Linie mit dem Modellelement verbunden werden, auf welches sich die Notiz bezieht.

2.1.5 Assoziationen

Eine Assoziation beschreibt die Relation zwischen Klassen und wird durch eine Linie zwischen zwei Klassen dargestellt. An den jeweiligen Enden kann die Vielfachheit der Beziehung angegeben werden. Die Vielfachheit einer Assoziation gibt an, mit wie vielen Objekten der gegenüberliegenden Klasse ein Objekt assoziiert sein kann. Jede Assoziation kann mit einem Namen versehen werden, der die Beziehung näher beschreibt. Damit man die Klassennamen und die Beziehungsamen in der richtigen Richtung lesen kann, kann neben dem Beziehungsamen ein kleines ausgefülltes Dreieck gezeichnet werden, dessen Spitze in die Leserichtung zeigt. Auf jeder Seite der Assoziation können Rollennamen dazu verwendet werden, um genauer zu beschreiben, welche Rolle die jeweiligen Objekte in der

Beziehung einnehmen. Außerdem können Zusicherungen verwendet werden, um die Beziehung speziell einzuschränken. In der folgenden Abbildung wird die Beziehung zwischen einer Firma und ihren Mitarbeitern gezeigt. Die Beziehung wird wie folgt gelesen: "1 Firma beschäftigt Mitarbeiter" Der Stern (*) steht für beliebig viele Instanzen.



Bild 2. Beispiel einer Assoziation[4]

2.1.6 Aggregation

Eine Aggregation ist eine Assoziation, die ein Ganzes-Teile-Hierarchie darstellt. Eine Aggregation soll beschreiben, wie sich etwas Ganzes aus seinen Teilen logisch zusammensetzt. Aggregationen sind gewöhnlich 1-zu-viele-Beziehungen. Eine Aggregation wird wie eine Assoziation als Linie zwischen zwei Klassen dargestellt und zusätzlich mit einer kleinen Raute versehen. Die Raute steht auf der Seite des Aggregats. Im Übrigen gelten alle Notationskonventionen der Assoziation. Das Beispiel Unternehmen-Abteilung-Mitarbeiter zeigt, dass ein Teil Abteilung gleichzeitig auch wieder Aggregat sein kann.

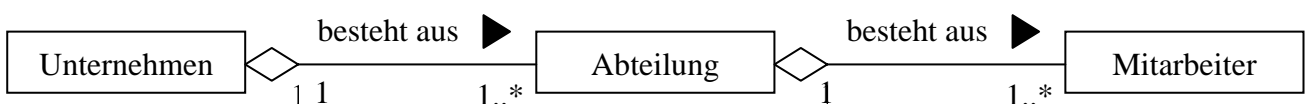


Bild 3. Beispiel einer Aggregation[4]

2.1.7 Komposition

Ein Komposition ist eine strenge Form der Aggregation, bei der die Teile vom Ganzen existenzabhängig sind. Das heißt, wenn das Ganze gelöscht wird, werden alle Einzelteile ebenfalls automatisch gelöscht. Wird Einzelteil gelöscht, bleibt das Aggregat erhalten. Die Komposition wird wie die Aggregation als Linie zwischen zwei Klassen gezeichnet und mit einer kleinen Raute auf der Seite des Ganzen versehen. Im Gegensatz zur Aggregation wird die Raute jedoch ausgefüllt. Hier ist ein Beispiel , eine Rechnung besteht aus

Rechnungspositionen. Wenn die Rechnung gelöscht wird, werden die Rechnungspositionen automatisch auch gelöscht.

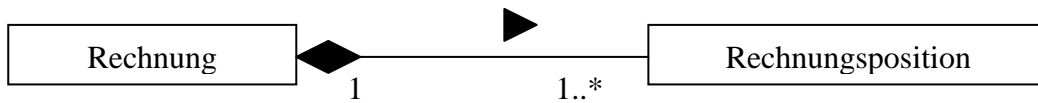


Bild 4. Beispiel einer Komposition[4]

2.1.8 Vererbung

Vererbung ist ein Mechanismus für die Relation zwischen Oberklasse und Unterklasse, wodurch Attribute und Operationen der Oberklasse auch den Unterklassen zugänglich werden. Mit Hilfe der Vererbung können Klassen hierarchisch strukturiert werden. Dabei werden Eigenschaften einer Oberklasse an die zugehörige Unterklasse weitergegeben. Die Vererbung wird mit einem großen, nicht ausgefüllten Pfeil, der von der Unterklasse zur Oberklasse zeigt dargestellt. Wahlweise werden die Pfeile direkt von den Unterklassen zur Oberklasse gezogen oder zu einer gemeinsamen Linie zusammengefasst. Die direkten Pfeile erlauben ein flexibleres Layout und sind auch mit freier Hand gut zu zeichnen. Die zusammengefassten Pfeile betonen stärker die Gemeinsamkeit der Unterklassen, nämlich dass sie Spezialisierungen einer Oberklasse aufgrund eines Diskriminators sind. Ein Beispiel: Ein Buch ist eine Oberklasse, ein englisches Buch, ein deutsches Buch und ein russisches Buch sind Unterklassen. Sie haben gemeinsame Diskriminators. Ein Roman und ein englisches Buch haben keine gemeinsame Diskriminators.

2.1.9 Abhängigkeit

Eine Abhängigkeit ist eine Beziehung zwischen zwei Modellelementen, die zeigt, dass eine Änderung in einem (unabhängigen) Element eine Änderung im anderen (abhängigen) Element notwendig macht. Beispiel hierfür ist : eine Klasse vererbt eine bestimmte Schnittstelle einer anderen Klasse. Wenn die angebotene Schnittstelle verändert wird, ist eine Änderung in der Klasse erforderlich. Dargestellt wird eine Abhängigkeit durch einen gestrichelten Pfeil, wobei der Pfeil vom abhängigen auf das unabhängige Element zeigt.

2.2 UML Profile

Ein UML Profile ist vordefinierter Satz von Constraints, Tagged Values und Stereotypen, um das UML Metamodell für eine spezielle Umgebung, einen Anwendungsbereich oder einen Prozess anzupassen.

Die drei Komponenten (Constraints, Tagged Values und Stereotypen) bildet das UML Profile. Das Konzept der Stereotypen ist ein Erweiterungsmechanismus in der UML , mit dem neue eigene Modellierungselemente auf dieser Basis definiert werden können. Auch Attribute, Operationen und Assoziationen können mit Stereotypen klassifiziert werden. Einer Stereotype wird in doppelte spitze Klammern geschlossen. Er wird jeweils über dem Elementnamen platziert. Tagged Values sind benutzerdefinierte, sprach- und werkzeugspezifische Schlüsselwörter, die Semantik einzelner Modellelemente um spezielle charakteristische Eigenschaften erweitern. Ein Constraint präzisiert oder beschränkt die Semantik eines beliebigen UML Modellelements. Ein Constraint ist ein Ausdruck, der die möglichen Inhalte, Zustände oder die Semantik einer Modellelementes einschränkt und der stets erfüllt sein muss. Constraint muss boolesche auswertbar sein. Im Modell werden sie in geschweifte Klammern gesetzt.[7]

3. Objekt-relationales Datenbank-Design mit der UML

Warum sollen wir den objekt-relationalen Datenbankenwurf durch UML repräsentieren?. Die beiden Techniken objekt-relationale Datenbank und UML sind objektorientiert. Ein ganzes System kann mehrerer Komponenten enthalten, z.B. Programmcode und Datenbank usw. Das ganze System, sowohl Programm als auch Datenbank, kann durch UML formuliert werden. UML ist eine erweiterbare Sprache, der man neue Modellelemente hinzufügen kann, um bestimmte Funktionen zu realisieren. Im folgenden werden ich versuchen eine Datenbank für die Verwaltung von Reservierungen eines Rechnerzimmers zu entwickeln. Ein UML Klassendiagramm wird als Werkzeug für den Konzeptuellen Entwurf benutzt und in ein logisches Schema (hier SQL:1999) umgewandelt. Bei der Implementierung wird Oracle8i eingesetzt.




	Database Element	UML Element	Stereotype	Icon
Architectural	Database	Component	<<Database>>	
	Schema	Package	<<Schema>>	
Conceptual	Persistent class	Class	<<Persistent>>	
	Multivalued Attribute	Attribute	<<MA>>	
	Calculated Attribute	Attribute	<<DA>>	
	Composed Attribute	Attribute	<<CA>>	
	Identifier	Attribute	<<ID>>	
Logical	Table	Class	<<Table>>	
	View	Class	<<View>>	
	Column	Attributes	<<Column>>	
	Primary Key	Attributes	<<PK>>	
	Foreign Key	Attributes	<<FK>>	
	NOT NULL Constraint	Attributes	<<NOT NULL>>	
	Unique Constraint	Attributes	<<Unique>>	
	Trigger	Constraint	<<Trigger>>	
	CHECK Constraint	Constraint	<<Check>>	
	Stored Procedure	Class	<<Stored Procedure>>	
Physical	Tablespace	Component	<<Tablespace>>	
	Index	Class	<<Index>>	

Tabelle 5. Stereotypen für Datenbank-Design[3]

In der Tabelle 5 werden die Stereotypen für den relationale Datenbank-Entwurf gezeigt. Sie werden gegliedert nach den verschiedenen Phasen des Entwurfs aufgeführt. Man beginnt beim Datenbankentwurf mit der konzeptuellen Modellierung. Es ist eine Abgrenzung eines Teils der „realen Welt“. Hier werden Stereotypen von einer Klasse und vier Attributen eingeführt. Eine Klasse in der Datenbank muss „persistent“ sein. MA, DA und CA sind neue Stereotypen für spezielle Attribute. ID steht für Identifikator. Die Beziehung zwischen Klassen kann durch gewöhnliche Assoziation der UML gezeichnet werden.

Nach der konzeptueller Modellierung kommt die logische Modellierung. Sie bezieht sich auf die logische Struktur der Datenbasis z.B. Tabellen, Attribute, Sichten. Die vorhandene Stereotypen sind an relationalen Datenbanken orientiert und bieten keine speziellen Stereotypen für das Objekt-relationale Modell, z.B. Stereotypen für Kollektion, Referenzen und Methode usw. Um das Objekt-relationale Modell durch UML logisch darzustellen, müssen neue Stereotypen eingeführt werden. Aber zuvor müssen wir die wichtigsten Komponenten des Objekt-relationalen Modells kennen lernen.

3.1 Die objekt-relationalen Modelle: SQL:1999 und Oracle8i

SQL:1999 ist momentan Standard für Objekt-relationale Datenbankmodelle und Oracle8i ist ein Beispiel für die Implementierung des objekt-relationalen Datenbankmodells. Einer der größten Unterschiede zwischen relationalem und Objekt-relationalem Modell liegt daran, dass das relationale Modell mindestens die erste Normalform hat. Das ist in den Basis-Regeln für relationales Design. Das objekt-relationale Modell ist jedoch nicht in der 1. Normalform. Eine Spalte von einer Objekt-Tabelle kann deswegen auch Kollektionstypen enthalten.

SQL1999 erlaubt noch neue strukturierte Datentypen für spezielle Applikation. Strukturierte Daten können als Attribute in einer Tabelle gespeichert werden. Strukturierte Daten können auch als ein Basistyp in der Definition einer Tabelle verwendet werden. Dies entspricht *Objekt Typ*. Die Tabelle ist eine Erweiterung von diesem Typen. In SQL1999 werden solche Tabellen auch *typed table* genannt. Wenn eine *typed Table* definiert ist, fügt das System eine neue Spalte für OID (Object Identifier) der Tabelle ein. Die Werte dieser Attribute werden von dem System generiert. OID ist eindeutig für jedes Objekt und kann vom Benutzer nicht geändert werden. Der Typ von OID ist ein Referenztyp (*REF*). Jeder Objekttyp hat unterschiedliche *REF* Typen. SQL1999 unterstützt einen anderen Strukturierten Datentyp, nämlich *ROW* Typ. *ROW* Typ hat festgesetzten Anzahl mit Elementen, die verschiedene Typen haben können. SQL1999 unterstützt nur einen Kollektionstyp *Array*. Im Vergleich zu SQL1999 hat Oracle8i auch die Typen: *Typed table*, *value type*, *REF* aber kein *ROW* Typ. Oracle8i unterstützt außerdem zwei Ableger des Kollektionstypen nämlich *VARRAY*, der äquivalent zu SQL1999 *Array* ist, und noch die verschachtelte Tabelle (*nested table*). Eine verschachtelte Tabelle ist eine Tabelle, in der eine Spalte wiederum eine Tabelle ist. Einer der größten Unterschiede ist, dass SQL1999 Vererbung unterstützt, aber Oracle8i nicht, weder Typen noch Tabellen.

3.2 Objekt-relationale Erweiterung für UML

Das objekt-relationale Modell hat viele Erweiterungen gegenüber dem relationalen Modell. Um ein objekt-relationales Schema durch UML zu repräsentieren, muss UML auch entsprechend erweitert werden. Dies kann durch ein UML Profile also Stereotypen, Tagged Values und Constraints ermöglicht werden. In Anhang A und Anhang B gibt es einen Überblick über die neue eingeführte Stereotypen für SQL:1999 und Oracle8i.

3.3 Methode für das objekt-relationale Datenbank-Design

Wir haben die neue Stereotypen und können jetzt objekt-relationale Datenbank durch UML repräsentieren. Wir empfehlen folgende Methode. Die Methode wird in die drei Phasen Analyse, Design und Implementierung unterteilt.

Analyse-Phase: Mittels UML Klassendiagramm kann ein konzeptuelles Schema entworfen werden.

Design-Phase: In diesem Phase gibt es zwei Unterphasen.

1. Standard-Design: Es ist ein logisches Design unabhängig von Produkten. Hier benutzen wir SQL1999.
2. Spezielles Design: Es ist ein Design für ein spezielles Produkt. Hier nehmen wir Oracle8i als Beispiel.

Implementierungsphase: In diesem Phase hat einige physikalische Designaufgaben. Das Ziel ist, die Antwortzeit und Speicherplatz zu verbessern.

UML	SQL:1999	Oracle8i
Class	Structured Type	Object Type
Class Extension	Typed Table	Table of Object Type
Attribute	Attribute	Attribute
Multivalued	ARRAY	VARRAY/Nested Table
Composed	ROW / Structured Type in column	Object Type in column
Calculated	Trigger/Method	Trigger/Method
Association		
One-To-One	REF/[REF]	REF/[REF]
One-To-Many	[REF]/[ARRAY]	[REF]/[Nested Table/VARRAY]
Many-To-Many	ARRAY/ARRAY	Nested Table/Nested Table VARRAY/VARRAY
Aggregation	ARRAY	Nested Table/VARRAY of References
Composition	ARRAY	Nested Table/VARRAY of Objects
Generalization	Types/Typed Tables	Oracle8i cannot directly represent the generalization concept

Bild 6 .Richtlinien für objekt-relationales Datenbank-Design[3]

3.4 Transformationsrichtlinien

Die Tabelle fasst ein paar Transformationsrichtlinien zusammen, die in der Methodologie verwendet werden können.

Transformation von Klassen, Attribute und Methode

Nur anhaltende Klasse müssen in Klassen des Datenbankschemas zu transformiert

werden. Eine anhaltende Klasse in UML ist markiert durch den Stereotyp <<persistent>>.

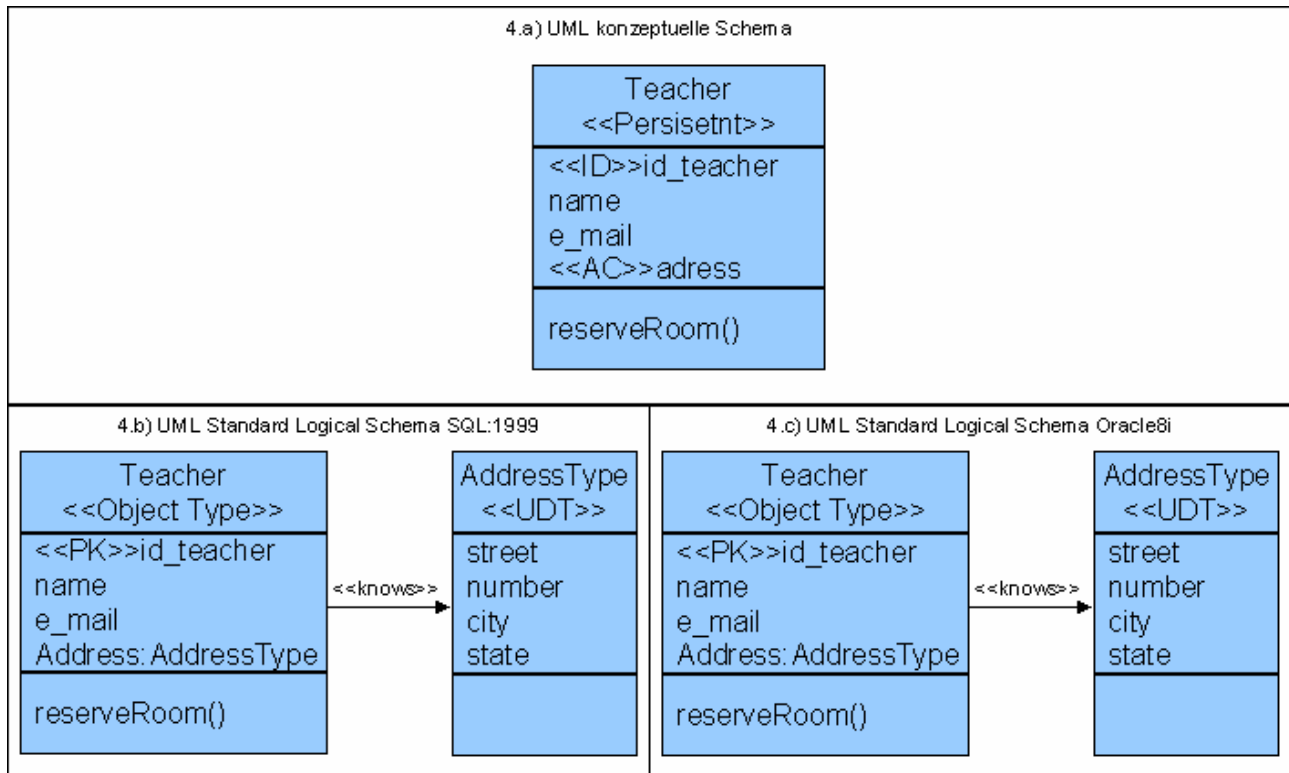


Bild 7. Transformation von Klassen, Attribute und Methode[3]

Hier ist ein einfaches Beispiel, wie eine anhaltende Klasse in SQL1999 und Oracle8i zu transformiert wird. Ein Lehrer hat vier Attribute, eine ID, name, e_mail und ein komponiertes Attribut address. Nach dem oben genannten Prinzip soll das Attribut address in SQL1999 und Oracle8i in ein Stereotyp <<UDT>> AddressType transformiert werden. In SQL1999 wäre es auch möglich in ROW Typ umzuwandeln. Die Klasse Teacher soll in SQL1999 und Oracle8i in ein <<Object Type>> transformiert werden. Das heißt, die Klasse Teacher soll in den beiden als eine strukturierte Tabelle repräsentiert werden.

Transformation von Assoziationen (Bidirektional)

Eine UML-Assoziation kann in einem objekt-relationalen Schema entweder als unidirektionale oder als bidirektionale Beziehung repräsentiert werden. Wir schlagen die folgende Transformationsregeln für bidirektionale Assoziationen vor. 1. One-To-one: Die Assoziation kann einfach in beide Objekt Typen durch ein Attribut mit REF Typ repräsentiert werden. 2. One-To-Many: Die Assoziation kann in einem Objekt durch ein Attribut mit REF Typ und in anderem Objekt-Typ durch Kollektionstyp repräsentiert werden. 3. Many-To-Many: Diese Assoziation kann in beider Objekt Typen durch ein Attribut mit Kollektionstyp dargestellt werden. In Oracle8i gibt es zwei Kollektionstypen. Falls die

Kardinalität bekannt ist, wird VARRAY empfohlen. Ansonsten soll die verschachtelte Tabelle verwendet werden.

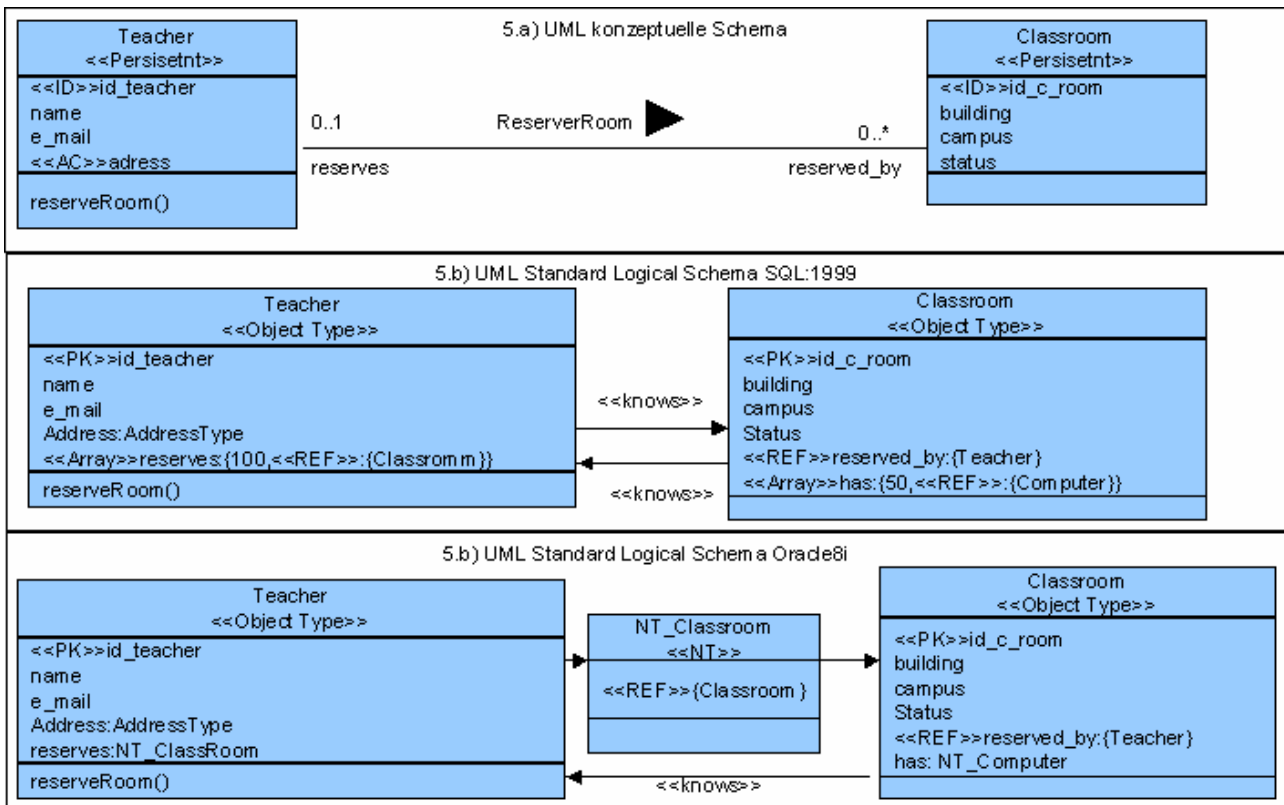


Bild 8. Assoziation Transformation[3]

Hier ist ein Beispiel für die Transformation einer One-To-Many Beziehung. Ein Lehrer kann mehrere Klassenzimmer reservieren. Aber ein Klassenzimmer kann hingegen nur von einem Lehrer reserviert werden. In SQL1999 kann diese Beziehung nur durch einen ARRAY-Typ repräsentiert werden. Also in die Tabelle Teacher wird ein Attribut namens reserves hinzugefügt. Das Attribut ist ein Array und hat maximal 100 Werte vom Typ REF. REF ist eine Referenz, die an ein Objekt von Classroom verweist. In der Tabelle Classroom wird ein Attribut reserved_by hinzugefügt, das ein Typ von REF ist. Das Attribut verweist auf ein Tupel der Tabelle Teacher. Oracle8i kann diese Beziehung entweder durch eine verschachtelte Tabelle oder durch ein VARRAY repräsentiert werden. In unserem Beispiel wird diese Beziehung durch ein NT (Verschachtelte Tabelle) repräsentiert. NT_Classroom ist die verschachtelte Tabelle, die nur eine Spalte von Typ REF hat. Hier gibt es keine Beschränkung über die Länge der verschachtelter Tabelle.

Transformation von Aggregation

Aggregation ist eine spezielle Art von Assoziation von Assoziation. Wir empfehlen, dass ein Attribut von Kollektionstyp in dem Ganzen zu definieren. Die Kollektion enthält die Referenzen, die an die Teile verweisen. In SQL1999 wird Kollektionstyp ARRAY benutzt. In Oracle8i kann VARRAY oder verschachtelte Tabelle benutzt werden. Komposition ist eine spezielle Aggregation. Die hat folgende Eigenschaft: Wenn das Ganze gelöscht ist, sollen die Teile vom Ganzen automatisch gelöscht werden. SQL1999 unterstützt dieses Verhältnis nicht. Dies kann aber durch Trigger realisiert werden. In Oracle8i kann dieses Verhältnis direkt durch verschachtelte Tabelle realisiert werden. In der verschachtelten Tabelle werden die Teile als Objekte gespeichert. Aber die so gespeicherten Objekte sind nicht wirklich Objekte. Sie haben keine OID und es kann nicht auf sie verwiesen werden.

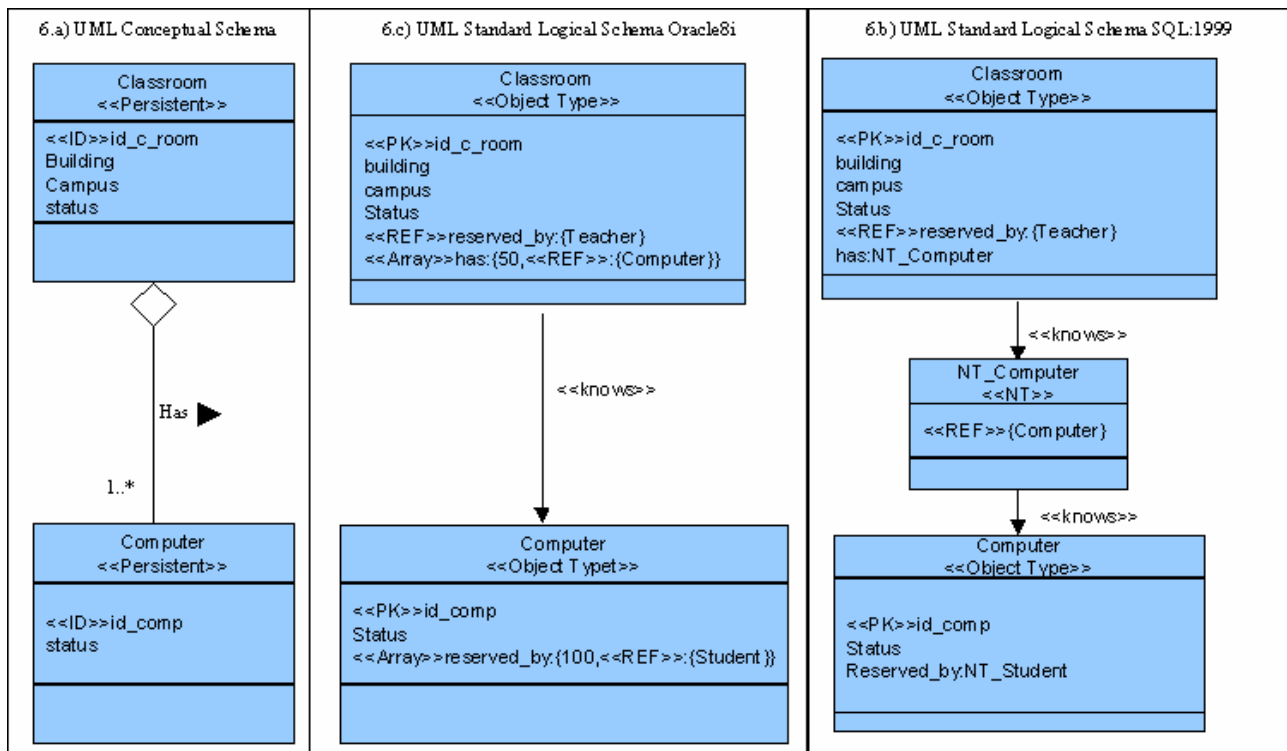


Bild 9. Aggregation Transformation[3]

Hier ist ein Beispiel für Aggregationstransformation. Ein *Classroom* hat mehrere *Computer*. In SQL1999 wird die Transformation durch ARRAY repräsentiert. Nach dem Beispiel hat ein *Classroom* maximal 50 *Computer*. Im ARRAY "has" wird die Referenzen an *Computer* gespeichert. In Oracle8i werden hingegen verschachtelte Tabellen verwendet. Die Referenzen zu *Computer* werden in einer Tabelle *NT_Computer* gespeichert. Es gibt keine Beschränkung für die Anzahl von *Computern*.

3.5 Semantik der UML

Dienen formale Sprachen in einem Informationssystem zur Repräsentation von Informationen, so bezeichnen wir die Sprache als Syntax. Die Bedeutung einer Sprache nennen wir Semantik [10]. Eine Semantik für UML heißt, dass die Bedeutung der Diagramme mit Hilfe einer formalen Sprache beschrieben werden. In unserem Beispiel ist die formale Sprache SQL. Durch die Transformationsregeln von UML-Klassendiagrammen zu SQL wird eine Semantik beschrieben.

3.6 Vergleich zwischen Semantik von SQL und der Superstructure

Das Ziel ist, herauszufinden, ob es Widersprüche zwischen Semantik von SQL und der Superstructure gibt. Dafür müssen wir die einzelne Modellelemente vergleichen.

- Operationen: In der Superstructure von UML 2.0 wird die Semantik von Operationen so beschrieben: "An operation is invoked on an instance of the classifier for which the operation is a feature." [9] Dies ist ein offensichtlicher Widerspruch, da Operationen in den SQL-Transformationen ignoriert werden. Ein Beispiel: Im Teacher-Beispiel gibt es Operation `reserveRoom()`, das nicht im SQL-Code auftaucht. Die Superstructure fordert aber, dass Operationen ausgeführt werden können.
- Assoziationen: In der Superstructure steht unter Semantics: "An association declares that there can be links between instances of the associated types. A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end." [9] Das stimmt mit den Transformationen überein. Eine Assoziation beschreibt eine Verbindung zwischen zwei Instanzen. Aber die Semantik von Assoziationen hat weitere Eigenschaft. "When one or more ends of the association have `isUnique=false`, it is possible to have several links associating the same set of instances. In such a case, links carry an additional identifier apart from their end values." [9] Das bedeutet, daß eine Instanz auch mehrmals mit einer anderen verbunden sein kann. Dies führt zu einem Widerspruch. Im Papier wird "isUnique" überhaupt nicht benutzt, aber auch nicht ausgeschlossen. Im Teacher-Beispiel kann ein Klassenzimmer nur von einem Lehrer reserviert werden

und ein Teacher kann mehreren Klassenzimmer reservieren.

- **Multiplizität:** In der Superstructure von UML 2.0 wird die Semantik von Multiplizität so beschrieben: „A multiplicity defines a set of integers that define valid cardinalities. Specifically, cardinality C is valid for multiplicity M if M.includesCardinality(C). A multiplicity is specified as an interval of integers starting with the lower bound and ending with the (possibly infinite) upper bound.”[9] Dies ist wieder ein offensichtlicher Widerspruch, da obere Grenze in den SQL-Transformationen ignoriert werden. Ein Beispiel: Im Beispiel Assoziation Transformation gibt es ein Attribut NT_ClassRoom, das ein Typ von verschachtelter Tabelle ist. Es gibt keine Beschränkung über die obere Grenze. Die Superstructure fordert aber, dass die obere Grenze beschränkt werden muss.
- **Attribute:** : In der Superstructure von UML 2.0 wird die Attribute so beschrieben: “Refers to all of the Properties that are direct attributes of the classifier. Subsets *Classifier::feature* and is a derived union. All instances of a classifier have values corresponding to the classifier’s attributes.” Es gibt einen Widerspruch. In den SQL-Transformationen hat eine Klasse die Attribute nicht nur über die Eigenschaften sondern auch die Beziehung mit anderer Klasse. Im Beispiel Assoziation Transformation gibt es ein Attribut reserves von Teacher. Das Attribut beschreibt eine Beziehung zwischen Teacher und Classroom.
- **Aggregation:** In Superstructure von UML 2.0 ist Aggregation ein Attribut von Property mit Typ Aggregationkind. „Aggregationkind is an enumeration type that specifies the literals for defining the kind of aggregation of a property. Precise semantics of shared aggregation varies by application area and modeler. The order and way in which part instances are created is not defined.” Das hat gleiche Semantik wie in den SQL-Transformationen.

4. Zusammenfassung

Die Methode für objekt-relacionales Datenbankdesign besteht aus drei Phasen. Also Analyse, Design, Implementierung. Das Klassendiagramm wird verwendet für konzeptuelles objekt-relacionales Datenbankdesign. Es ist vieler geeignet als E/R-Diagramm. Denn UML ist wegen der Erweiterbarkeit und Objektorientierung besonders. geeignet für objekt-relacionales Datenbankdesign. Mit Hilfe der Methode, kann

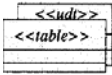

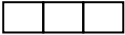

eine Datenbank leicht entworfen werden.

5. Literaturen

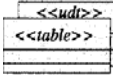

1. **Margarete Payer, Alois Payer** : Datenbankaufbau. Grundkonzepte von Datenbanken <http://www.payer.de/dbaufbau/dbauf01.html>
2. **Kemper, Eickler**: Datenbanksysteme Eine Einführung, 4. Auflage **Oldenbourg, 2001**
3. **Esperanza Marcos, Beln Vela, and Jos Mara Caverro**: A Methodological Approach for Object-Relational Database Design using UML. *Software and Systems Modeling*, 2(1):59-72, 2003
4. **Bernd Oestereich**: Objekt-orientierte Software-entwicklung Analyse und Design mit der UML 2.0, 6. Auflage. **Oldenbourg, 2004**
5. **Bernd Brügge, Allen H.Dutoit**: Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java, 2. Auflage. **Pearson Studium, 2004**
6. UML-Tutorial <http://ivs.cs.uni-magdeburg.de/~dumke/UML/>
7. **Tom Pender**: UML Bible. **Wiley Publishing, 2004**
8. **Marc Born, Eckhardt Holz, Olaf Kath**: Softwareentwicklung mit UML 2. **ADDISON-WESLEY,2004**
9. **UML 2.0 Superstructure Specification, OMG Final Adopted Specification ptc/03-08-02**

10. Manfred Broy: Informatik: Eine grundlegende Einführung Teil I,
2.Auflage. Springer Berlin, 2000

Anhang A

SQL:1999 Stereotypes	
<p>Structured Type Metamodell Klasse: Klasse Beschreibung: ein <<UDT>> erlaubt die Repräsentation von neuen benutzer-definierten Datentypen. Icon: Nein Zusicherung: nur benutzt bei Definition von Value Typen Tagged values: Nein</p>	<p>Typed Table Metamodell Klasse: Klasse Beschreibung: definiert als <<Objekt Type>>. Eine Tabelle eines strukturierten Datentyps Icon:  Zusicherung: impliziert die Definition eines strukturierten Typs Tagged values: Nein</p>
<p>Knows Metamodel Klasse: Assoziation Beschreibung: <<knows>> verbindet eine Klasse mit einem Benutzer definierten Datentyp <<UDT>>. Eine uni-direktionale Relation. Die Richtung wird durch einen Pfeil angegeben und endet bei definiertem Datentyp. Icon: Nein Zusicherung: nur Verbindung einer Klasse mit <<UDT>> Type Tagged values: Nein</p>	<p>REF Type Metamodel Klasse: Attribute Beschreibung: repräsentiert ein Link zu einer <<Object Type>> Klasse. Icon:  Zusicherung: Ein <<REF>> Attribut kann nur ein <<Object type>> referenzieren. Tagged values: die <<Object type>> Klasse</p>
<p>Array Metamodel Klasse: Attribute Beschreibung: repräsentiert einen indizierte Kollektionstypen mit fester Länge. Icon:  Zusicherung: beliebiger Typ außer <<Array>> Tagged values: der Basistyp von array die Länge der Elemente</p>	<p>Row Type Metamodel Klasse: Attribute Beschreibung: repräsentiert ein komponiertes Attribut mit fester Anzahl von Elementen. Die können verschiedene Datentypen haben. Icon:  Zusicherung: keine Methode Tagged values: Der Name der Elemente und Datentypen</p>
<p>Redefined Method Metamodel Klasse: Methode Beschreibung: <<redef>> ist eine vererbte Methode, die von der Unterklasse noch einmal implementiert wird. Icon: Nein Zusicherung: Nein Tagged values: Parameter, Type und Rückgabewert.</p>	<p>Deferred method Metamodel Klasse: Methode Beschreibung: ein <<def>> Methode. Implementierung erste in unterer Klasse. Icon: Nein Zusicherung: muss in obere Klasse definiert werden. Tagged values: Parameter, type und Rückgabewert.</p>

Anhang B

Oracle8i Stereotypes	
<p>Object Type Metamodel Klasse: Klasse Beschreibung: ein <<UDT>> erlaubt die Repräsentation von neuen Benutzer definierten Datentypen. Icon: Nein Zusicherung: nur benutzt bei Definition von Value Typen Tagged values: Nein</p>	<p>Object Table Metamodel Klasse: klasse Beschreibung: definiert als <<Objekt Type>> . Ein Tabelle eines strukturierten Datentyps.  Icon: Zusicherung: impliziert die Definition von eines structured type Tagged values: Nein</p>
<p>Knows Metamodel Klasse: Assoziation Beschreibung: verbindet eine Klasse mit einem Benutzer definierten Datentyp <<UDT>>, <<Array>> oder <<NT>>. Eine uni-direktionale Relation. Die Richtung wird durch einen Pfeil angegeben und endet beim definierte Datentyp Icon: Nein Zusicherung: nur Verbindung eine Klasse mit <<UDT>> Type Tagged values: Nein</p>	<p>REF Type Beschreibung: repräsentiert ein Link zu <<Object Type>> Klasse Icon: ●→ Zusicherung: Ein <<REF>> Attribut kann nur ein <<Object type>> referenzieren Tagged values: die <<Object type>> Klasse</p>
<p>VARRAY Metamodel Klasse: Attribute/Klasse Beschreibung: repräsentiert eine indizierte und fest lange Kollektion Type. Icon: Nein Zusicherung: beliebiger Typ außer <<Array>> und <<NT>> Tagged values: der Basistyp von array die Länge der Elemente</p>	<p>Nested Table Metamodel Klasse: Klasse Beschreibung: <<NT>> repräsentiert einen nicht-indizierten Kollektionstyp ohne feste Länge  Icon: Zusicherung: beliebige Type außer <<Array>> oder <<NT>> Tagged values: Der Basistyp der verschachtelten Tabelle</p>

